Improvement QoE-Driven Application Deployment and Energy Module Arrangement in Fog Environments with Offloading

Low Choon Keat, Chew Zheng Hing, Ng Yen Phing and Tew Yiqi

Tunku Abdul Rahman University Management and Technologies, Malaysia

DOI: https://dx.doi.org/10.47772/IJRISS.2025.9010293

Received: 13 January 2025; Accepted: 17 January 2025; Published: 20 February 2025

ABSTRACT

The Internet of Things (IoT) and other rapidly evolving technologies have profoundly affected daily life and created an exponential rise in the amount of data generated and processed. By extending cloud capabilities to the network edge, fog computing lowers latency and boosts the effectiveness of data processing. But it also brings with it new difficulties, especially regarding resource management and energy usage. This study starts with a thorough analysis of the current state of fog computing systems, pointing out weaknesses and areas for improvement. We suggest enhancing current QoE-Aware application allocation algorithm, energy-aware module allocation methods, and task-offloading approaches to maximize resource efficiency in light of our research. Experiments in simulated fog computing environments are used to assess these methods, with an emphasis on performance measures including energy-aware module allocation metrics, QoE-aware application allocation enhancement, and offloading applications developing.

Keywords: Edge Computing, Fog Computing, Internet-of-Things, Internet-of-Things, Quality-of-Experience, Energy Comsumption

INTRODUCTION

Over the past few decades, technology has evolved at an unprecedented rate, reshaping the way we live and work. One of the most transformative advancements is the Internet of Things (IoT), which has become an essential part of our daily lives. “Thing” in IoT can refer to any item that possesses the necessary processing power, Internet connectivity, and network collection and transmission capabilities without aid or manual intervention. For example, the automobile equipped with sensors that can transmit a real-time alert about any malfunction, a human with an implanted health monitor, an animal farm with transponders in every animal, or anything in the world with an IP address and the capacity to transfer data over the internet can all be considered "things" in the context of the Internet of Things and all these devices equipped with advanced processors and ample memory which can help in handle multiple tasks efficiently. However, in order to provide the efficiency and stability that IoT promise, cloud computing plays a vital role. It can be shown the cloud computing are able to offer a robust infrastructure that supports the massive amounts of data generated by IoT devices and it provides us the scalable storage solutions and computational power, which are essential to help in processing and analyzing data in real-time. Thus, this capability ensures that IoT applications can function smoothly without the need for extensive on-premises hardware.

Despite these advantages, cloud computing is not without its limitations, particularly in the context of Industry 4.0. According to Potu et al. (2022), issues such as unstable internet connections and limited bandwidth pose significant obstacles to efficient data transmission, real-time processing, and analysis. Additionally, industrial sensors and controllers often require more computing power than cloud systems can directly support, and security and privacy concerns add further complications.

To address these challenges, fog computing has emerged as a viable solution, bridging the gap between cloud servers and IoT devices. As noted by Rahimikhanghah et al., (2021), fog computing introduces an intermediate layer that brings computing and storage resources closer to the end users. This proximity helps alleviate network congestion and reduce latency, providing more efficient data processing. However, fog computing is not without its own challenges. For example, the process of managing a large number of fog nodes can be complex, and issues related to Quality of Experience (QoE), energy consumption, and task offloading need careful consideration for a lot of reasons.

It can be demonstrated that a low Quality of Experience (QoE) will significantly contribute to user dissatisfaction. Therefore, the QoE-driven application deployment are aims to enhance the overall user experience by optimizing the placement of applications based on the capabilities of fog nodes and user requirements (Chen et al., 2020). By improving QoE, we can ensure that users receive timely and reliable services, which is essential for maintaining the efficiency and effectiveness of IoT systems.

Next, energy consumption in fog computing is a significant concern, as the increased number of fog nodes can lead to higher energy usage. Optimizing energy consumption involves dynamically adjusting the processing power of fog nodes to match the demands of tasks, thereby reducing unnecessary energy expenditure. This not only helps in lowering operational costs but also contributes to environmental sustainability by minimizing the carbon footprint of computing activities.

Besides, computation offloading plays a vital role in preventing the overloading of individual fog nodes by strategically redistributing tasks from heavily loaded nodes to those with available resources, computation offloading ensures a balanced workload across the network. This approach helps in maintaining the performance and reliability of fog computing systems, allowing for more efficient data processing and resource utilization (Sheikh Sofla et al., 2021).

In response to these issues, research has focused on developing effective strategies for task scheduling and resource management in fog computing environments. By implementing QoE-driven application deployment policies, optimizing energy consumption, and employing computation offloading techniques, we can enhance various performance metrics such as latency, cost, and energy efficiency. These approaches aim to improve user service quality, minimize data processing times, and reduce network congestion.

This project explores improvement of existing solutions to enhance fog computing performance, focusing on QoE-aware application deployment, energy-efficient module arrangement, and strategic computation offloading. The goal is to ensure a seamless and efficient integration of fog computing within IoT ecosystems, ultimately providing better services to end users. Lastly, this research aims to advance the capabilities of fog computing, making it a more reliable and effective component of the modern technological landscape by addressing these critical aspects.

Background

Fog computing, also known as edge computing, is a decentralized computing infrastructure that brings computational resources and services closer to the data sources, at the edge of the network. This approach addresses the limitations of traditional cloud computing, especially in scenarios requiring real-time data processing and low-latency responses.

Then, fog computing architecture is generally divided into three layers. The first layer, known as the IoT Devices Layer (End Tier), includes various Internet of Things (IoT) devices such as sensors, actuators, and user devices. These devices generate and collect data from their environment, serving as the entry point for data into the fog computing system. The second layer, termed the Fog Layer, acts as an intermediary and consists of fog nodes or gateways equipped with storage, computing, and networking capabilities. These fog nodes perform local data processing, storage, and preliminary analysis. Examples of fog nodes include routers, switches, gateways, and even devices like surveillance cameras and micro data centers'. By handling tasks that require quick response times, the fog layer reduces the amount of data that needs to be sent to the cloud. At the top of this architecture is the Cloud Layer, which consists of centralized cloud data centers'. This layer is responsible for extensive data storage, aggregation, and complex data analysis, leveraging the massive computational power and scalability of cloud resources (Charaf et al., 2021).

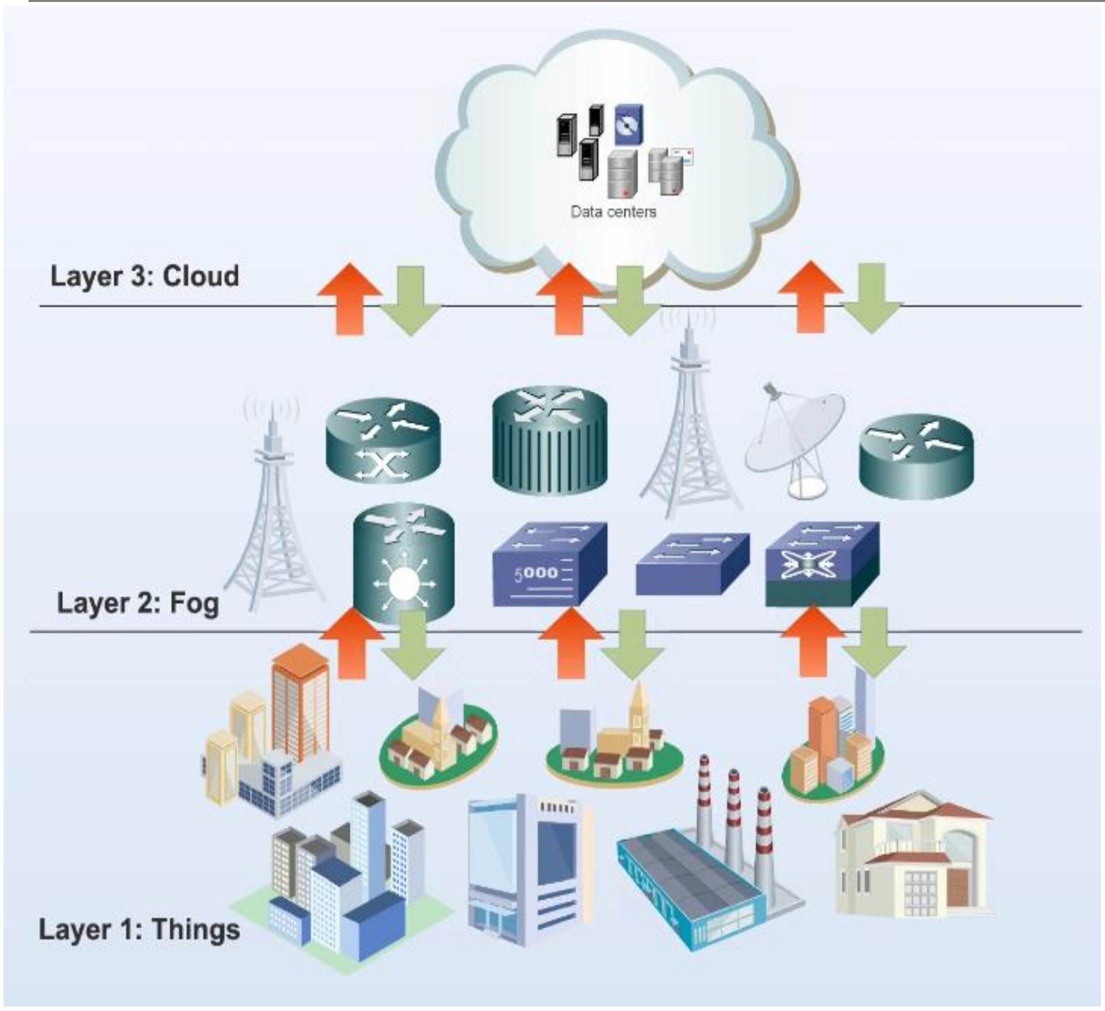


Fig. 1. Fog cloud Architecture

Advantages & contributions

Fog computing offers several significant advantages that can greatly enhance the performance and efficiency of various applications and systems (Abdali et al., 2021).

One key advantage is its ability to reduce latency. By processing data closer to the source, fog computing minimizes the physical distance data must travel. This proximity enables quicker data processing and reduces latency, which is crucial for applications requiring real-time responses, such as online gaming, video conferencing, and VoIP. With data being processed locally, these applications can achieve faster response times and improved user experiences.

Another important benefit is bandwidth efficiency. Local data processing at fog nodes means that less data needs to be transmitted to the cloud. This reduction in data transmission alleviates network congestion and optimizes bandwidth usage. As a result, network resources are used more efficiently, which is particularly beneficial in large-scale IoT deployments where vast amounts of data are generated.

Enhanced security is also a significant advantage of fog computing. Since data is processed locally at fog nodes, the exposure of sensitive information during transmission to distant cloud servers is reduced. This localized processing can implement security measures closer to the data source, thereby enhancing the overall security of the system and reducing the risk of data breaches.

Moreover, fog computing contributes to energy efficiency. Fog nodes can offload energy-intensive tasks from resource-constrained IoT devices. By handling these tasks locally, fog nodes help optimize energy consumption and can extend the battery life of IoT devices. This is particularly important for devices that operate in remote or inaccessible locations where battery replacement or recharging is challenging.

Therefore, the advantages of fog computing become even more apparent when considering its role in addressing the limitations of traditional cloud environments. The diagram in Figure 1.1 illustrates the growing trend of organizations embracing multi-cloud architectures. According to the Flexera 2023 State of the Cloud Report, 87% of organizations are adopting multi-cloud strategies, combining various cloud services to meet their diverse needs (Luxner, 2023).

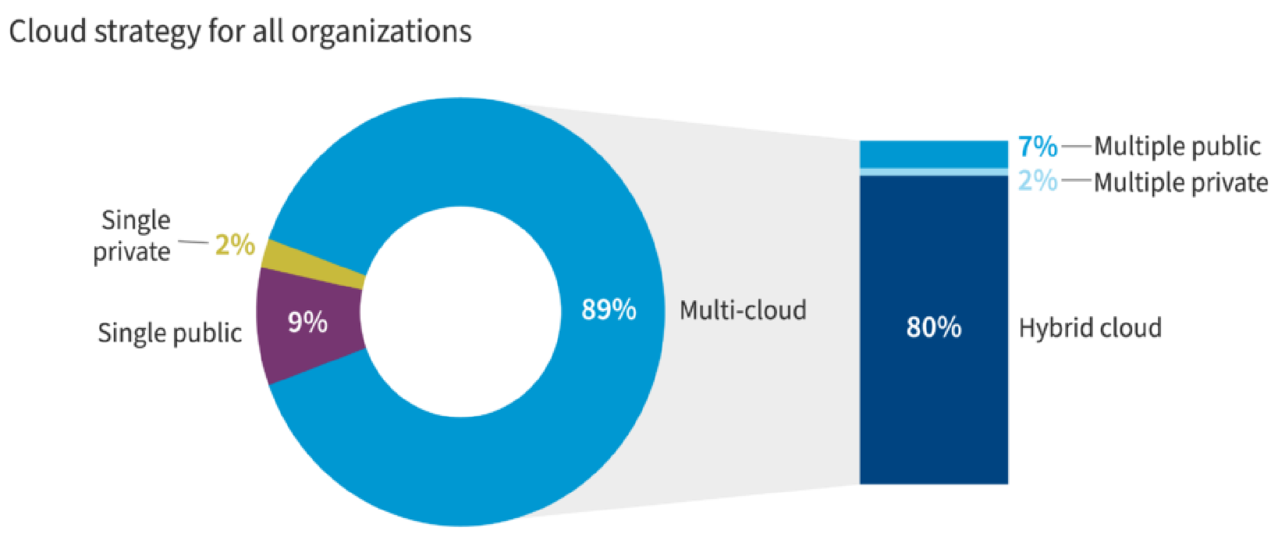


Fig. 1.2 Usage of Fog cloud in 2023

This widespread adoption of multi-cloud architectures demonstrates the shift towards more flexible and efficient computing solutions. Multi-cloud environments typically involve a combination of public, private, and hybrid clouds, allowing organizations to optimize their operations by leveraging the strengths of each cloud type. In this context, fog computing plays a crucial role in enhancing the overall efficiency and performance of these multi-cloud setups.

Research Motivation

With the rapid expansion of the Internet of Things (IoT), the demand for cloud computing has surged, reflecting its integral role in handling the vast amounts of data generated by countless IoT devices. As more sensors and smart devices come online, they continuously collect and transmit data to cloud servers, enabling a wide range of applications from smart homes to industrial automation. However, this escalating demand has revealed significant challenges within cloud computing infrastructures (Swarnakar et al., 2023). Issues such as increased latency, network congestion, and bandwidth limitations have become critical concerns, highlighting the need for more efficient and responsive solutions.

Fog computing has emerged as a promising approach to address these limitations by extending cloud capabilities to the edge of the network. This intermediary layer, situated between cloud data centers and IoT devices, brings computational resources closer to where data is generated. Current fog computing technologies leverage advanced techniques such as edge analytics, real-time data processing, and decentralized storage to mitigate the drawbacks of centralized cloud computing. For instance, the implementation of fog computing in smart grid systems has enhanced energy management by enabling real-time monitoring and control. Additionally, autonomous vehicles rely on fog computing to process data locally, reducing latency and improving safety.

The adoption of fog computing is gaining momentum across various sectors. According to Figure 1.3, the global fog computing market is expected to grow from USD 162.3 million in 2022 to USD 9698.2 million by 2032, driven by the increasing deployment of IoT devices including software and hardware (Fog Computing Market Size, Share | CAGR of 52.1%, n.d.). In healthcare, fog computing is utilized to support telemedicine services and patient monitoring systems, ensuring timely and reliable data transmission. Similarly, in smart cities, fog computing facilitates efficient traffic management, environmental monitoring, and public safety initiatives.

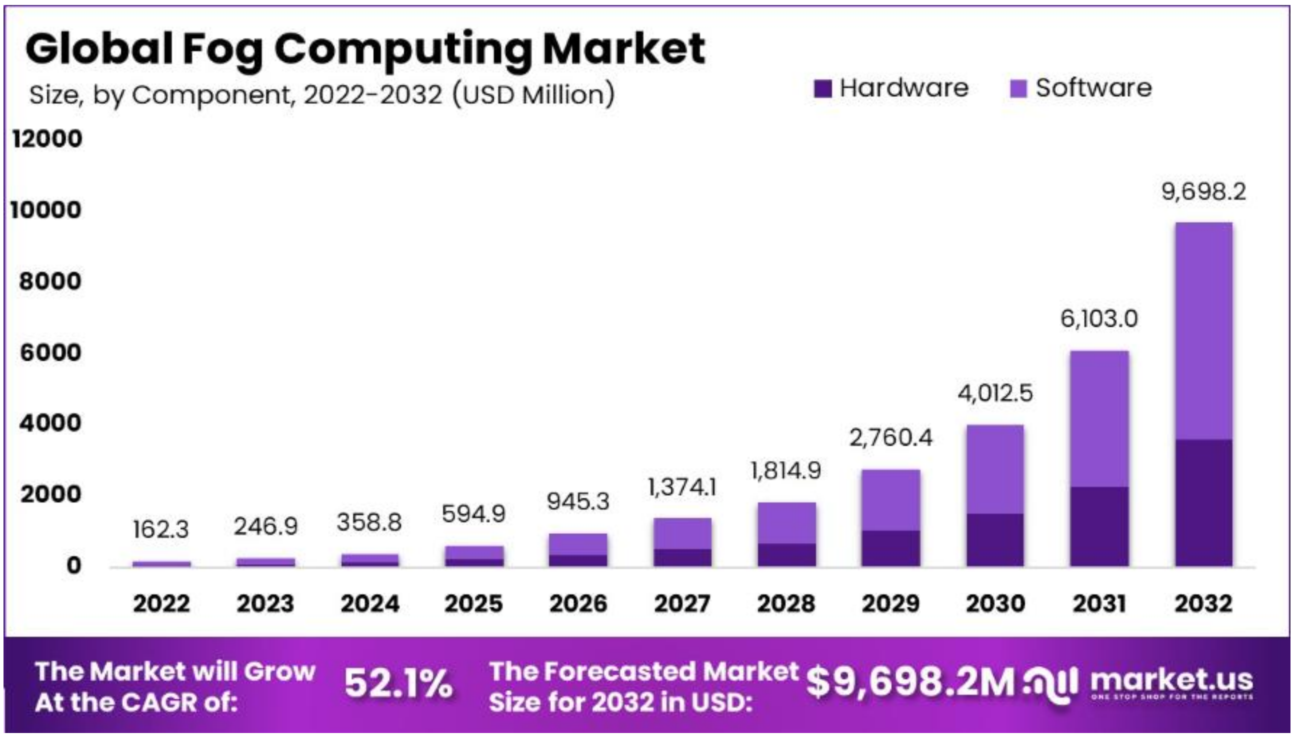


Fig. 1.3 Global Fog Computing market from year 2022-2032

Our motivation for this research is to explore ways to enhance current fog computing technologies. By focusing on Quality of Experience (QoE)-driven application deployment and energy-efficient module arrangement, coupled with strategic task offloading, we aim to optimize the performance and sustainability of fog computing systems. The effective adoption of these techniques will assure fog computing service providers a competitive advantage and operational sustainability, allowing them to meet the increasing demands of modern applications. This will have a significant impact on various industries, including smart cities, healthcare, and industrial IoT, ensuring continuous, efficient, and tailored service delivery.

By advancing fog computing technologies, we hope to contribute to the development of more resilient and adaptive systems that can better support the dynamic needs of today's digital world. This research aims to not only improve technological capabilities but also deliver tangible benefits to society by enhancing the efficiency and effectiveness of critical services.

Statement of the problem

Fog computing, as an extension of cloud computing, promises to address some of the inherent limitations of the cloud, such as high latency and network congestion, by bringing data processing closer to the edge devices. This architecture, designed to bridge the gap between cloud data centers and edge devices, offers significant improvements for latency-sensitive Internet of Things (IoT) applications, including healthcare services and real-time analytics. Despite these advantages, fog computing present's new challenges, particularly in terms of energy consumption and resource management (Mostafa, 2020).

The primary problem addressed in this project is the high energy consumption and sustainability concerns associated with current fog computing architectures. With the growing proliferation of IoT devices and the increasing demand for real-time data processing, there is a critical need for energy-efficient scheduling and offloading strategies. Modern server hardware in fog environments often consumes a substantial amount of power, even when idle, contributing to increased operational costs and environmental impact. Moreover, the dynamic and distributed nature of fog computing complicates the allocation of computing resources, making it challenging to maintain high Quality of Experience (QoE) for end-users.

Statement of the Objectives

Our project aims to develop advanced offloading strategies to maximize Quality of Experience (QoE) and energy efficiency in fog computing environments. Our objectives is to:

1. To find the way of improve the existing QoE-aware application mapping policy for enhancing the user satisfaction.
2. Design and implement energy-efficient module arrangement strategies in fog environments to minimize energy consumption.
3. Develop and implement effective task offloading strategies in fog computing to optimize resource utilization and improve overall system performance.
4. To evaluate the suggested solution and compare the work with the existing solutions.

Scope of the Research

This research focuses on enhancing the performance and efficiency of fog computing systems through the improvement of QoE-driven application deployment, energy-efficient module arrangement, and effective task offloading strategies. The scope of this study includes:

1. Comprehensive Understanding:

The research will start with a comprehensive review of existing solutions and approaches in fog computing and include an analysis of current challenges, limitations, and opportunities for improvement in QoE, energy efficiency, and task offloading for the basic understanding.

2. Review of Existing Solutions:

A thorough review of existing solutions will be conducted to identify gaps and areas for enhancement. This will involve analyzing the effectiveness and shortcomings of current approaches in addressing QoE, energy consumption, and resource management issues in fog computing.

3. Development of Enhancement Mechanisms:

Based on the review and analysis, the research will develop enhancement mechanisms for QoE-driven application deployment, energy-efficient module arrangement, and effective task offloading in fog computing systems. These mechanisms will aim to address the identified gaps and improve the overall performance and efficiency of fog computing systems.

4. Experimental Evaluation:

The proposed mechanisms will be evaluated through experiments in simulated fog computing environments. This will include implementing the mechanisms and conducting tests to assess their effectiveness in improving QoE, reducing energy consumption, and optimizing resource utilization.

5. Performance Metrics:

The evaluation will focus on performance metrics such as QoE improvement, energy consumption reduction, and system performance enhancement. These metrics will be used to quantify the impact of the proposed mechanisms and compare them with existing solutions.

Chapter summary & evaluation

Our research focuses on enhancing the performance and efficiency of fog computing systems through the improvement of QoE-driven application deployment, energy-efficient module arrangement, and effective task offloading strategies. The chapter begins by discussing the evolution of technology, particularly the Internet of Things (IoT), which has become integral to modern life. As the number of IoT devices grows, cloud computing faces challenges such as increased latency and network congestion. Fog computing emerges as a solution, offering local processing and networking functions to reduce latency and improve data processing efficiency.

The research motivation stems from the escalating demand for cloud computing due to the proliferation of IoT devices, highlighting the need for more efficient and responsive solutions. Fog computing addresses these challenges by extending cloud capabilities to the edge of the network. However, fog computing presents new challenges, particularly in terms of energy consumption and resource management.

The primary problem addressed in this project is the high energy consumption and sustainability concerns associated with current fog computing architectures. This problem is exacerbated by the growing number of IoT devices and the increasing demand for real-time data processing. To address these challenges, the research aims to improve the existing QoE-aware application mapping policy, design energy-efficient module arrangement strategies, and develop effective task offloading strategies in fog computing.

The scope of the research includes a comprehensive understanding of fog computing, a review of existing solutions, and the development of enhancement mechanisms. Experimental evaluation will be conducted to assess the effectiveness of the proposed strategies in improving QoE, reducing energy consumption, and optimizing resource utilization. Performance metrics such as QoE improvement and energy consumption reduction will be used to evaluate the proposed mechanisms.

In summary, our research seeks to advance fog computing technologies to provide better services to end users and meet the increasing demands of modern applications. By enhancing QoE-driven application deployment, energy-efficient module arrangement, and effective task offloading strategies, we aim to optimize the performance and sustainability of fog computing systems.

LITERATURE REVIEW

There are several primary fog computing challenges such as placement, energy and offloading to be focused on and reviewed in this section. There are various relevant research papers that have been analyzed and reviewed for the purpose of achieving high efficiency in module placement, optimization of energy consumption, and great offloading performance in the past. Next, based on these various research papers, there are some obstacles and shortcomings found in the fog-cloud environment. In short, a new algorithm is able to be proposed for solving the obstacles and shortcomings.

There are four sections in this chapter which include the project background, literature review, QoE placement, Energy-aware, task offloading and a conclusion.

Project Background

The technologies to be focused on in this chapter are the Internet of Things (IoT) in which is evolving at a rapid speed and is treated as a crucial source of big data. According to (He, 2020), IoT is embedded in a network and an example of an IoT object can include electronics hardware, software, sensors and network connectivity. IoT also further allows data to be gathered and interchanged by these physical objects. IoT consists of three crucial parts that play an important role in completing the whole picture of IoT. The parts of “things” (objects), the part of the computer system that uses data streaming to and from objects and the part of communication networks that is used to form a connection between them. Based on Xu et.al (2022), the rise of the IoT as the main connectivity medium for billions of industrial sensors, smart home appliances, and consumer wearable devices is paving the way for novel communication and computation paradigms that can assist in scaling such unexpected new communications. The realization of ICT is brought through algorithms and applications based on different aspects.

Research by Raza (2020) stated that cloud computing is able to handle a large volume of storage without affecting the performance of the service. Besides, according to Yan et.al (2022), cloud computing has been recognized as a paradigm for big data storage, analytics and it offers better solutions for the implementation of loT applications by developing its degree to manage things in a distributed environment. Therefore, IoT with cloud computing has emerged in industry and life. According to Chen. et.al., (2020), the combination of cloud computing and IoT enables widespread sensing services and powerful processing of sensing data streams. Significant advances in core technologies, processing, and communication algorithms are leading to new intelligent IoT services in our lives such as smart cities, smart industries, smart grids, smart healthcare, and so on to improve all aspects of life.

On-demand self-service, widespread network access, resource pooling, quick elasticity, and measured service are attributes of the cloud. Despite its advantages (cost savings, efficiency, scalability, and reliability), cloud computing faces significant difficulties when dealing with large amounts of data. Additionally, because the cloud is a centralized computing paradigm, the majority of computations take place there directly. However, the current cloud computing services still contain limitations with large amounts of data transferring which will cause latency and limited resources to be computed at the same time. In other words, Cloud Computing has the limitations of ultra-low latency, high bandwidth, security, and real-time analytics.

Thus, the survey by Costa et.al. (2022) stated that fog computing was proposed to overcome the limitations of cloud computing by bringing the computation closer to the edge of the network. The purpose of fog computing is to process what is to be done by the cloud in the fog layer to reduce the workload between the end devices and the cloud centre. Manzoor et.al. (2022) demonstrated that the fog nodes are located close to the end users and these nodes offer resources such as computing, storage, and networking to the applications operating under this infrastructure unlike cloud computing which has a centralized data centre to manage resources. It is thus very important to make reasonable resource scheduling decisions to ensure the quality of service and reduce resource waste. Fog computing provides advantages due to the geographically distributed fod nodes, real-time data processing and low latency. With these characteristics, fog computing is suitable to be deployed for applications that are very sensitive to delay. The applications can include smart cities, smart vehicles, smart traffic lights, etc. While the fog provides localization, enabling real time interaction and low latency at the network edge, the cloud provides centralization, the integration of which inspires applications that require the interplay and cooperation between the edge (fog) and the core (cloud), particularly for big data and the Internet of Things. Moreover, an emerging wave of Internet deployments, which is the Internet of Things (IoTs), requires mobility support and low latency and a new platform is needed to meet these requirements which is fog computing (Abkenar et.al., 2022). The coming trend in the world will be the Internet of Things (IoTs) that use the internet and smart devices in our daily life such as connected vehicles, smart grid, smart campuses, smart homes and wireless sensor and actuator networks. Therefore, our potential markets can be the users of IoTs and the users of cloud computing.

Review result

Closed to User

The centralization of the Cloud data centers has caused some drawbacks in the Cloud-IoT integration. The difference is that in the Fog Computing environment, Fog services hosted on Fog Nodes (FNs), are not only toward the network edge but also distributed everywhere along the continuum from the Cloud-IoT. Any device can become a fog node as long as it has enough storage, computing, and networking resources to process advanced services. Therefore, fog nodes can be either (i) end devices with rich resources (for example, smart traffic lights, vehicles, video surveillance cameras and industrial controllers) (ii) edge nodes (for example, switches, wireless access points and cellular base stations) and (iii) specialized “core” network routers.

Fog computing overcomes the latency issue in Cloud Computing by carrying out data analytics near the source where data is collected so that the response times become predictable. This is an important attribute for lots of IoT applications. Besides that, since the fog nodes are located closer to IoT devices, it helps to overcome the limitation of context awareness. This is because exploiting context information allows service improvement and resource utilization optimization.

The amount of data exchanged and transmitted with a Cloud data centre can be reduced as some portion of the data is communicated with nearby fog nodes which act as agents between the IoT and the Cloud. Therefore, the volume of Big Data can be efficiently processed, thus reducing bandwidth consumption.

Fog computing has the attribute of providing improved privacy and security in IoT applications and addressing security issues. This is because the fog node locally stores and analyzes the sensitive data stored and only allows the Cloud to access part of the sensitive data. In this scenario, Fog will process data for privacy enforcement that is not applicable for resource-constrained IoT devices (such as the extraction and transmission of metadata, complex encryptions). Fog Computing helps to provide connection availability even in a Hostile environment. When IoT devices experience no connectivity or discontinuous connectivity to the Cloud, a nearby fog node is able to provide the IoT device with critical services. **Sample Heading (Third Level).** Only two levels of headings should be numbered. Lower level headings remain unnumbered; they are formatted as run-in headings.

System-level Paradigm

Fog Computing is a system-level paradigm, where a single resource-rich computer could not provide the overall service; this is because the service is decomposed and provided by different fog nodes which process a specific service at the same time cooperating with other fog nodes (Shaifali et al, 2021). One of the guiding principles of the OpenFog Reference Architecture (OFRA) is the pyramid-like organisation as shown in Figure 2.1.

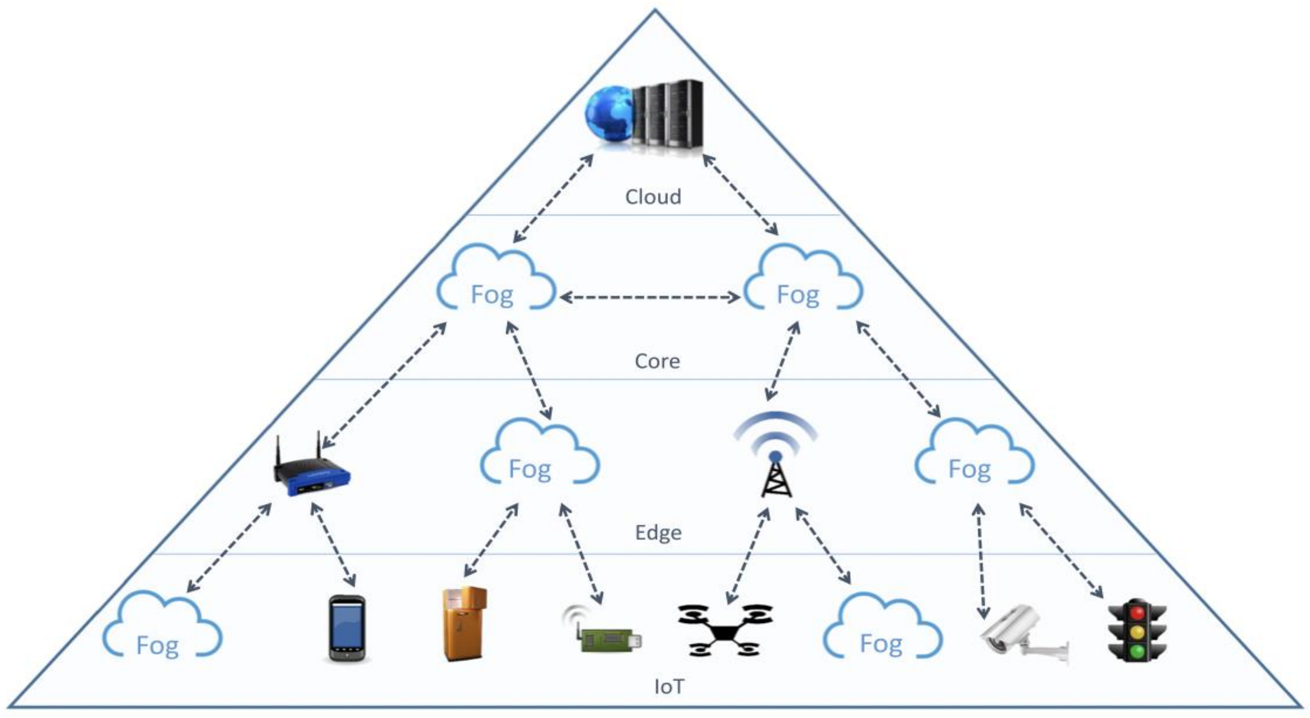


Fig. 2.1 Fog Computing pyramid-like hierarchical organization

The lowest layer hierarchy contains the IoT (end devices or Things), in which rich-resources IoT might themselves act as fog nodes. The higher layers of the hierarchy numbers and composition depend on the actual application domain and purpose from the network edge up to the core. The Cloud is at the highest layer. Fog Computing is the expansion of Cloud in providing services within the same layer or among nodes belonging to different layers. The role of each fog node depends on its position in the pyramid.

In short, fog computing can be said to consist of three main layers, IoT devices layer (End Tier), Fog layer (Fog Tier), and Cloud layer (Cloud Tier). The IoT devices layer is the layer where the sensors gather all the users' requests from the application. The fog layer behaves as intermediary between IoT end devices layer and cloud layer and helps in QoE, efficiency energy consumption and calculation offloading. Fog nodes are devices with computing capability, storage and network connectivity such as switches, routers and video surveillance cameras. Cloud layer is the highest layer in this fog computing architecture which receives information from fog nodes and then conducts analysis.

The Examples of Application of Fog Computing

Smart Utility Service

The primary goal of smart utility services is to reduce costs and save time by saving energy. In order to enable analysis of data from the application at every minute for real time update and to address the difficulty in transmitting other data-heavy traffic caused by IoT applications, fog computing is advantageous.

Smart Cities

Traffic regulation is one of the most important uses of fog computing in smart cities. To gather information on how vehicles move on the road, sensors are embedded in the traffic lights and road barriers. Real-time data analysis made possible by fog computing enables the traffic signal to change quickly in response to the flow of traffic.

Healthcare

The evolution of wearables is introduced by technological progress and IoT. From a watch that tells the time and date to a smartwatch that does more than that by providing users with other data, such as their health status. The wearables are also used on hospital patients to continuously provide information on their vital signs, blood sugar levels, and other things. These devices benefit from fog computing because it guarantees timely data delivery in emergency situations.

Video Surveillance

In order to give video footage of public behaviour, surveillance cameras are typically mounted in shopping malls and other public areas. A large amount of data is gathered by surveillance cameras in the form of video. Fog computing is crucial in identifying anomalies in crowd dynamics and promptly alerting authorities to the issue in order to avoid lag.

Autonomous Vehicles

Autonomous vehicles rely on rapid data processing for navigation and safety. Fog nodes within the vehicle or nearby infrastructure process data from various sensors, enabling real-time decision-making. This minimizes latency and ensures the vehicle can respond swiftly to changing conditions.t the top, the Cloud layer includes centralized data centers and cloud servers with extensive computing and storage capabilities. It receives processed information from fog nodes for further analysis and long-term storage, focusing on large-scale data management and complex analytics. The hierarchical organization of fog computing allows efficient data management, reduces latency, and enhances real-time processing, making it a scalable and effective solution for handling the massive data generated by IoT devices.

Advantages and Limitations in Fog Computing

Fog computing utilises computing components at the network edge to serve as an intermediary layer between end devices and Cloud data centres. Fog nodes are computing components such as computers, Raspberry Pi, micro-data centres, and gateways used in fog environments.

Delayed Consciousness

Fog computing utilises computing components at the network edge to serve as an intermediary layer between end devices and Cloud data centres. Fog nodes are computing components such as computers, Raspberry Pi, micro-data centres, and gateways used in fog environments.

Location Awareness

Most IoT apps are context-aware, which means they prepare themselves based on the surrounding environment and other apps. Sending all of these context-aware application queries to the cloud isn't realistic (Bridges et.al., 2020) but realistic if sent to a fog device.

Limitation of Fog computing

The limitation of fog computing in QoE is that the different applications have different application placement policies in order to achieve a certain service level. Furthermore, QoE is frequently changing so proposing a variety of real-time requirements for fog environments.

Introduction of fog computing does successfully solve the problem of cloud such as delay issue and location awareness issue. However, it greatly increases the energy consumption in the computing environment which contrasts with the “green computing” concept.

Fog computing is facing difficulty in evenly distributing tasks to different fog devices. This issue will cause some of the fog devices to face overhead problems while some fog devices are having nothing to handle.

As a result, solving the QoE, energy and offloading limitations in the fog computing are required. Hence, understanding what other people have researched on QoE, energy and offloading are important.

QoE Placement, Energy Aware, Task Offloading

Quality of experience (QoE) Review

The QoE is primarily connected to the general customer satisfaction level with a vendor. Additionally, QoE can be applied to any customer-related service or business and is frequently used in information technology (IT) and consumer electronics. Nevertheless, the evolution of IoT devices has required the IoT devices to exchange data or information between themselves and also with the cloud data centre. The obstacle is that the cloud data centre or cloud storage does not have enough computational resources to support this huge amount of information, which will lead to latency issues. As a result of latency issues, the QoE delivered by cloud computing will be decreased (Saovapakhiran et.al., 2022). Hence, this is the reason why fog computing is necessary for extending cloud computing.

Fog computing is a decentralized computing infrastructure, where the location of data, computers, storage, and applications is in between the data source (IoT devices) and the cloud. By doing so, fog computing could execute some workload of cloud computing because of its structure, which could decrease the length of transmission of data and the overall bandwidth needed (Bartosz Kopras et al., 2022). To fulfill QoE for cloud services, fog computing has been developed. Cloud storage is utilized in many real-world applications, such as cloud gaming, video and image processing which require weighty processing power to analyze and evaluate data in a limited time. Fog computing offers essential services such as processing data locally instead of in the cloud. Consequently, there will be a decrease in latency and network bandwidth as a huge amount of time-sensitive data from applications or IoT devices is processed locally.

In short, as justified by Mazur et al (2021), QoE is an important measurement of the level of customer satisfaction in order to retain them. QoE is an assessment of customers including expectations, perceptions, cognition, feelings and satisfaction when prescribing a service, application or product. Hence, it can be said that QoE assists fog computing in enhancing user satisfaction when using a service.

QoE Related Work

The placement policy of QoE-aware applications introduced by Mahmud et al (2020) contains various fuzzy logic-based methods that prioritize many application placement requests and categorize fog computational instances in line with user expectations and the instances' current status, respectively. To allow low latency response requirements in IoT applications, Cloud-like services will be provided by fog computing at the network edge. The reason that application deployment in fog is a burdensome problem is because of the nature of computing instances which are hierarchical, dispersed, and heterogeneous. Each of the fog nodes has distinctive network round-trip time, data processing speed, and resource accessibility which cause the difficulty in putting applications in fog. However, to meet certain service level targets, different application placement policies are mandatory in fog. Therefore, application placement in fog has utilized QoS, resource, and situation-aware features. Additionally, in the computer environment, the deployment of applications to applicable fog instances is able to fully utilize the user QoE in resource usage, utility access, and service delivery (Laghari et al., 2023). Based on Mahmud et al (2020) experimental findings, this approach can significantly minimize data processing times, network congestion, resource costs, and service quality.

By using modified TOPSIS, Baranwal et al. (2020) have come up with a lightweight QoE aware application placement policy in fog computing. According to Gaurav and his team's relative presumption and computational capabilities for the placement policy, this approach precedes the applications and fog instances. Applications with distinctive values of dependent metrics were found to receive the same precedence when fuzzy logic was deployed in previous research. Fog computational instances may have the same precedence even with numerous levels of computing competence. However, the downside of this approach is that it requires complex computing. The modified TOPSIS not only inherits all the strengths of the classical TOPSIS but also removes the rank reversal problems which successfully get rid of the downsides of the approach. For comparative research, simulation experiments indicate that the proposed model greatly lessens the time for application placement as compared to the state-of-the-art. In the meantime achieving the targeted resource usage, processing time, and reduced network congestion.

Based on the Multi-Dimensional QoE (MD-QoE) model, H.Nashaat et al. (2020) have introduced an IoT application placement technique. Conventional methods are used to define the user QoE expectations of subjectively evaluating QoE, for instance, feedback-based approaches like Mean Opinion Scores (MOS) and Net Promoter Score (NPS). Nevertheless, the previous assessment methodologies might not be suitable for the IoT environment because they measure QoE subjectively. For example, real-time events frequently occur in the IoT, delivering feedback at each dedicated interval to govern QoE has led to an increase in network latency and slower application response times. To satisfy the QoE influence factors (IFs), resource distribution of application placement requests is required in fog environments. The approach has been divided into two main sections. In the first stage, divergent IoT application placement requests are prioritized based on the 3 main IFs which are (i) environment runtime context, (ii) application use, and (iii) user expectations. Feedback will be given through QoS violations. In the second stage, mapping and routing the request to the suitable fog node instance depends on its position, processing capability, and expected response time. In short, the proposed technique has enhanced overall system performance but will increase power consumption.

A resource management strategy has been created in the Varshney et al. (2021) study that implements the AHP technique and controls the various Fog resources by assessing the values of chosen QoE criteria in the Fog computing environment. The suggested strategy takes into account an experimental study for analyzing the various outcomes. The suggested experiment's performance is assessed based on QoE metrics such as network bandwidth, typical latency, storage capacity, and processing speed. The fog computing environment can allocate resources to smart applications in accordance with their needs.

In Zhao et .al (2021) paper, they proposed a QoE-driven cross-layer optimization scheme for secure video transmission over the backhaul links in cloud-edge networks. They developed a secure transmission model based on video encoding and edge caching. They formulated a joint optimization problem of video encoding parameters and an edge caching strategy to improve QoE. Then, a near-optimal algorithm was designed to solve the joint optimization problem. Furthermore, a greedy algorithm with low complexity to obtain the suboptimal solution was proposed too and has proven to improve video encoding quality and reduce transmission latency and be more robust for caching capacity and could ensure secure transmission for more videos with the limited caching capacity of edge caching servers.

Wang et.al (2022) have proposed a QoE metric that integrates the bitrate of a tile’s representation, the relationship between the tile and the user's viewport, the user's distance to the tile, the occlusion between tiles and the resolution of the display screen based on perspective projection in projective geometry. Furthermore, they have developed a greedy-based rate adaptation algorithm. They have demonstrated that their proposed solution has near-optimal performance with low execution time, has outperformed existing tile-based algorithms and non-tiling schemes in transmission efficiency and achieved the highest peak-signal-to-noise ratio (PSNR) under limited bandwidth, whereas other non-tiling approaches held the highest PSNR for a sufficiently large bandwidth.

Saovapakhiran et.al. (2022) reviewed QoE metrics and QoE optimization objectives for various kinds of problems such as prediction models, optimization and control, and resource management respectively. The paper categorized emerging IoT architecture problems as QoE-aware offloading problems, QoE-aware placement problems and QoE-aware data caching problems. Their paper discovered that ML-based approaches were used to predict QoE and solve resource allocation problems which requires a novel concept of AI-based layers for managing QoE. The drawback of the proposed model based on QoE is that it does not take completion time into consideration.

Sreenivasu Mirampalli et al. (2022) proposed a resource allocation strategy for fog-enabled mission-critical IoT applications using the Hungarian Maximization Algorithm and a fuzzy-based approach. This method optimizes the matching of IoT applications to fog instances, maximizing user Quality of Experience (QoE) while meeting Quality of Service (QoS) constraints. The findings indicate that this approach significantly reduces data processing times, network congestion, and resource costs, enhancing overall service quality.

According to Mirzapour-Moshizi and Sattari-Naeini (2022) paper, they proposed a QoE-aware application placement framework for fog computing environments using the Simple Additive Weighting (SAW) method combined with game theory. This framework aims to optimize the placement of IoT applications by considering user expectations and various QoE parameters, such as processing speed, proximity to gateways, and cost. The proposed approach divides the fog environment into multiple domains, each managed by a gateway that oversees several nodes. Using game theory and the SAW method, the framework determines the most suitable domain to handle each application, and then applies the Particle Swarm Optimization (PSO) algorithm to select the most appropriate node within that domain. Simulations conducted in iFogSim show that the framework significantly reduces service response times compared to existing methods, particularly for real-time applications, while also improving resource utilization and reducing network congestion. By integrating user expectations into the placement process, the framework enhances the overall Quality of Experience (QoE) for users. The study concludes that the proposed method outperforms traditional approaches in terms of performance, response times, and resource efficiency, making it a robust solution for IoT application placement in fog environments. Future research will focus on incorporating mobility and additional parameters to further enhance the framework's performance in more dynamic settings.

Z. Yang et al. (2022) presents a QoE-aware task processing controller based on fuzzy logic to optimize task allocation in IoT edge computing systems. This mechanism efficiently manages task distribution by considering multiple QoE and QoS parameters, such as network congestion, resource availability, and service quality, to prioritize and allocate tasks effectively. The controller uses fuzzy logic to make informed decisions about task placement, enhancing resource utilization and network performance. Simulations conducted in iFog-Sim show that the proposed mechanism significantly reduces network congestion, minimizes latency, and lowers energy consumption at the IoT network edge, making it more efficient for handling tasks with limited resources. Overall, the fuzzy-based approach outperforms traditional methods, proving to be an effective solution for managing task allocation in IoT environments.

Yadav & Baranwal (2023) introduced a novel QoE-aware mechanism based on fuzzy logic for task allocation in IoT edge computing systems. The research focuses on optimizing network usage, reducing latency, and lowering energy consumption by considering multiple QoE parameters in task allocation decisions. The proposed fuzzy task allocation mechanism effectively improves resource management and network performance in IoT edge environments.

Carvalho & Macedo (2023) propose a QoE-aware container scheduling algorithm that extends the Kubernetes scheduler to improve user experience in cloud environments. The key contribution of their research is the use of deep learning models, specifically Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), to estimate the Quality of Experience (QoE) that the cloud can offer. The proposed algorithm monitors cloud resource usage and employs these estimations to schedule and reschedule containers based on QoE objectives. Experimental results demonstrate that their scheduler improves average QoE by at least 61.5% compared to other schedulers, and the proposed rescheduling method enhances QoE by up to 119%. The evaluation considered two QoE-aware applications: live classroom and video on demand. This study highlights the significant impact of integrating QoE objectives into container scheduling to enhance user experience.

Evangeline et al. (2023) proposed a fault-tolerant multimedia cloud framework to ensure Quality of Experience (QoE) in live streaming. The key contribution of the research is to address issues related to resource allocation, bandwidth sharing, and fault tolerance in a cloud multimedia environment. The proposed framework incorporates novel algorithms for efficient queuing, resource allocation, bandwidth allocation, and fault tolerance to guarantee the desired level of QoE. Their findings suggest that the proposed approach significantly improves the prediction accuracy and fault tolerance in live streaming applications.

Feng et al., (2023) propose a framework that utilizes digital twin technology to optimize resource allocation, including model selection, transmit power, computation time, and GPU-cycle frequency. The study specifically employs the generalized fractional programming theory, Lagrangian dual decomposition, and an adaptive modified harmony search algorithm to solve these optimization problems, ensuring fairness in QoE across users. The results show that these algorithms effectively balance the QoE of worst-case clients, improving overall system performance Jain & Kumar, (2023) used to optimize Quality of Experience (QoE) by addressing resource utilization and task offloading in fog computing environments. The paper formulates the task offloading problem as a Markov Decision Process (MDP) and employs Deep Reinforcement Learning (DRL) methods like Deep Q-Network (DQN), Deep Deterministic Policy Gradient (DDPG), and Soft Actor-Critic (SAC). These methods aim to maximize resource utility, balance service latency, energy consumption, and ensure task deadlines and priorities are met, thus improving QoE in fog environments.

Hosseinzadeh et al., (2023) discusses the use of control-theoretic approaches, specifically model predictive control (MPC), to optimize QoE in video streaming by managing bandwidth allocation among multiple competing video players. The CANE framework focuses on improving QoE fairness by considering the player's algorithm, state, and overall network conditions. It uses machine learning techniques to model the behavior of video players and allocates bandwidth to balance both efficiency and fairness across players, demonstrating significant improvements in QoE fairness over client-side adaptive bitrate (ABR) algorithms.

Ghasemi (2024) introduces the Multi-Objective Harris Hawks Optimization (MOHHO) algorithm for service placement in fog computing environments, aiming to optimize the placement of services by balancing multiple objectives, specifically reducing end-to-end delay and energy consumption. The algorithm addresses the challenges of multi-objective optimization by converting them into single-objective problems, using two sets of solutions: one focused on minimizing delay and the other on minimizing energy consumption. The best solutions from each set are then compared using non-dominant sorting, and the optimal global solution is selected. Simulation results in the CloudSim environment show that the proposed MOHHO algorithm outperforms existing methods such as Random Mapping, Genetic Algorithm (GA), Modified Genetic Algorithm and Particle Swarm Optimization (MGAPSO), and Teaching Learning-Based Optimization (TLBO). It achieved up to 44% reduction in energy consumption compared to random mapping and 14% less than TLBO, reduced end-to-end delay by up to 34% compared to random mapping and 10% compared to TLBO, and improved network utilization efficiency by up to 43% compared to random mapping and 12% compared to TLBO. Although the execution time of MOHHO is slightly higher than GA, it is lower than MGAPSO and TLBO due to the complexities involved in optimization. The study concludes that the MOHHO algorithm effectively optimizes service placement in fog computing, enhancing energy efficiency and reducing latency. It demonstrates superior performance compared to current algorithms and has potential applications in other cloud computing challenges like scheduling and load balancing. Future work will explore integrating additional meta-heuristic algorithms to further improve performance in dynamic computing environments.

Abofathi et al. (2024) introduced a novel method for optimizing service placement in fog computing environments using a Distributed Learning Automata (DLA) algorithm. The study focuses on improving energy consumption and delay in IoT applications by efficiently distributing modules across fog nodes. Results indicate that the DLA-FMP algorithm outperforms other optimization techniques in terms of energy efficiency and delay reduction.

Bikas and Sayıt (2024) proposed a genetic algorithm-based path selection approach for Multipath TCP (MPTCP) to maximize the Quality of Experience (QoE) in adaptive HTTP streaming systems. The key contribution of the research is to jointly consider bandwidth and delay differences, as well as the disjointness of paths, in the selection process. The findings suggest that this approach significantly improves QoE metrics compared to methods that consider bandwidth or delay differences individually.

Liu et al. (2024) propose a novel framework for QoE-aware collaborative edge caching and computing tailored for adaptive video streaming. The primary contribution of this research is to optimize video quality and minimize latency by intelligently distributing computational and caching tasks across edge devices. The findings demonstrate that the proposed approach significantly enhances user experience by reducing buffering time and improving video quality.

Islam et al. (2024) introduced the HPSP algorithm design, utilizing Tabu Search Algorithm as a meta-heuristic to improve QoE over POPP by proactively deploying service instances. The study analyzed QoE and deployment cost, showing that QoE enhances with HPSP due to the Hyper-Heuristic strategy, while deployment cost decreases with task size but rises with the number of ENs. The optimization formulation focused on QoE-based optimization for service instance deployment, reducing it to an MKP problem and considering user QoE and deployment costs in the MEC environment.

Our proposed QoE-aware application placement policy for Fog differs from the aforementioned works since we have considered multiple user expectation parameters such as service access, resource requirement, processing time, response rate, resource availability, processing speed, decentralized management, prioritized placement, deadline and compound QoE gain. The policy is developed in a decentralized manner so that it is less susceptible to single points of failure and management overheads. Application placement and met requests are prioritized based on users' expectations and the compound QoE of users is maximized through the policy.

Energy Aware Review

To enhance fog computing performance, load balancing and processing power are shared among fog nodes. For boosting performance, offloading has to take into consideration energy usage, task load, waiting time and network situations. As a result to design an energy-saving compute offloading strategy according to deep learning that can fulfill the optimum offloading selection. Even though the aforesaid technique might greatly enhance latency and energy consumption, the problems of data transmission and processing security are not included in the aforesaid solution.

Bichi et.al. (2022), proposed that the Internet of Things enables sensors and devices to examine their surroundings and make autonomous decisions. Innovative surveillance technologies have cleared the ground for the military-based IoT to emerge. However, these high-security conditions are harsh and unpredictable, and the devices deployed in such situations must operate constantly for an extended period of time. The new IoT-Fog architecture paradigms, which encompass key sectors, are becoming possible for real-time decision making, and optimal energy conservation in IoT-based application data processing is crucial. As a result, it is critical to select an effective application architecture for operation that may conserve energy over the application's lifetime.

(Malik et al., 2022) stated that offloading tasks saves energy. However, if task offloading is used in conjunction with device control-based energy saving strategies, the amount of energy saved can be increased even further. These strategies govern some functionalities or features of devices in order to improve performance and conserve energy. Local devices and task helper nodes can use device control to change parameters such as transmission power, on/off switching time, battery supply voltage, battery supply frequency, and modulation scheme.

One key purpose of offloading is to lower the energy consumption of mobile devices when performing tasks that require more energy. A mobile gadget that tries to handle everything on its own may quickly exhaust its battery due to space limits. As a result, while passing tasks to the server for execution, performance in terms of energy usage must be addressed. (Chuang & Hsiang, 2022)

Energy Consumption Work

Two semi-greedy based algorithms, (i) priority-aware semi-greedy (PSG) and (ii) PSG with multi start process (PSG-M) were proposed by Azizi et al. (2022) to map IoT tasks to fog nodes effectively. They have also developed task scheduling to satisfy the need for QoS in IoT jobs while at the same time reducing the overall energy consumption of fog nodes. The primary purpose was to meet the job deadlines while fully utilizing the overall energy usage of the system. If it has exceeded a job's deadline, it is dedicated to a fog node that enables the slightest deviation from the job's time limit requirements. The two effective precedence aware semi-greedy algorithms are recommended to meet these objectives. Extensive experiments are used to assess the efficiency of the suggested algorithms. The findings describe that the proposed algorithms greatly suppress previous algorithms in terms of the proportion of IoT tasks that met their time limit requirements, the overall energy consumption and system lifespan, and the aggregate amount of the deadline violation time.

A scheduling algorithm is a technique that assigns jobs to available system resources (Nazari Bu-Ali et al., 2022). It should be noted that incompatible scheduling algorithms may result in hardware inefficiencies or application slowdowns. Using a suitable algorithm, on the other hand, reduces energy usage and response time. The recommended solution employs a genetic algorithm with Non-dominated sorting and a Simulated Annealing algorithm. The research's primary innovations and points are described as (i) obtaining priority tasks in the form of a DAG graph, with graph construction and weighting determined by network communication and transmission latency, and dynamically allocating priority jobs. (ii) Create inventive first solutions for the suggested algorithm. (iii) Consider how communication delays affect system response times. (iv) Using a multiobjective algorithm to optimize both energy usage and reaction time. (v) Select the best option from the list of possibilities using the DVFS method.

Naha et al.(2022) answered a problem statement and proposed energy-aware resource allocation to make the Fog environment sustainable when satisfying time sensitive application requirements while available resources in the devices are changing dynamically. Naha and the team determine which resources are suited for energy-aware resource allocation. As a result, multiple linear regression is used to govern application execution in an energy-conscious manner. The linear regression-based strategy is used to construct an energy-aware resource allocation algorithm and to determine how all the independent factors affect the dependent variables using linear regression.

In the paper by (Mordacchini et al., 2022), it offers a decentralized, self-organizing, and QoE-centric scheme for optimizing the system's energy consumption. The technique allows Edge entities to interact with one another in order to exchange information and decide whether the users of each application may be served with fewer instances. This behavior allows you to limit the number of instances running in the system, which saves energy and resources. When deciding whether to shut down a potentially redundant instance, the entities use the data they have communicated to determine whether this decision is consistent with the QoE of the services and the computational limits of Edge resources. The simulation results suggest that the proposed method can lower the energy consumed by the system by about 40%.

In this paper by (Feng et al., 2022), it presents a novel transmission strategy-based NOMA transmission in a multi-IoT cooperative fog computing system with one task node and numerous nearby fog nodes. To reduce energy consumption, processing tasks from the task node can be offloaded to IoT devices using partial offloading. To formulate an energy consumption minimization problem for the complete IoT fog computing system, the study will consider fog node selection, duration allocation, offloading workload, and local computation resources. In addition, to acquire the fog node selection, the paper will reformulate an assignment problem by constructing a bipartite graph. Due to the coupling of local computing resource allocation and offloading burden, it divided the origin nonconvex problem into two comparable subproblems and presented the MCTC algorithm to solve it. To obtain the closed-expression solution of computation time and local CPU frequency, solve the first subproblem. The second subproblem can then be proven to be convex and effectively addressed. The simulation results demonstrate that the proposed algorithm outperforms other benchmark techniques by at least 56.88%.

In the paper by (Delgado & Famaey, 2022), it is shown that energy-aware scheduling mechanisms are needed to improve the performance of successful application execution on batteryless devices. These tiny gadgets often turn on and off, so knowing how much energy will be spent and how much energy may be recovered is critical. As a result, the paper presents theoretical insights into the attainable performance increase of energy-aware task scheduling when compared to state-of-the-art non-aware batteryless application task schedulers in this study. Furthermore, as a first step toward constructing a workable scheduling heuristic that can run on batteryless devices, the paper also investigates the effect of the size of the look-ahead energy prediction window. To accomplish this, Delgado & Famaey proposed a new optimal energy-aware scheduling algorithm that takes into account the energy available in the capacitor as well as the expected energy to be harvested in order to optimally schedule the tasks, which are defined by their priority, arrival time, execution time, energy consumption, and set of task parents that must be executed beforehand.

The article by (Avgeris et al., 2022) introduced the ENERDGE framework, which addresses full task offloading and resource allocation issues in a multi-site environment. Avgeris suggested a holistic energy-aware resource optimization method based on VM flavor design and supplemented with a unique load redistribution technique based on MRFs, together with the ultimate goal of minimizing total energy usage without losing QoS in terms of latency. ENERDGE examines the dynamic wireless conditions of the access network and enables a mobility prediction system to better guide the allocation solution during task offloading to minimize the inverse impact of dynamic user presence. The prediction mechanism accurately forecasts users' mobile behavior, according to numerical studies, and the ENERDGE resource optimizer surpasses two well-established load balancing algorithms in terms of latency and energy usage. Finally, the article demonstrated that the MRF technique rapidly converges to minimal energy solutions, allowing for efficient future energy optimizations.

Using Deep Reinforcement Learning, (Sellami et al., 2022) demonstrated the feasibility of establishing task assignment and scheduling algorithms for SDN-enabled IoT networks. It is devised as a task assignment and scheduling issue that reduces network latency while maintaining energy efficiency. The solution outperforms deterministic placement algorithms, random algorithms, and A3C strategies in determining optimal allocation decision policies for task assignments and scheduling in real-time. In addition, Sellami et al. technique enabled both local and global optimization, resulting in lower-latency communication and increased energy efficiency. Thus, it claims that it can extend the DRL method to provide intelligent multi-access Ultra-Dense Edge Computing (UDEC) to more efficiently utilize multiple 5G resources.

The protocol for Energy-efficient Fog Computing-enabled Data Transmission (EFoCoD) in Tactile Internet-based Applications is proposed by (Idrees et al., 2022). In the Tactile Internet-based fog computing architecture, the protocol operates at the sensor device level. To decrease data reading redundancy in this device, the EFoCoD protocol employs the LiDaRE algorithm at the sensor devices. Several studies have been carried out to demonstrate the efficacy of the proposed strategy. When compared to PFF and ATP, the EFoCoD protocol reduces the quantity of transferred data and reduces the sensor device's energy usage from 87.23% to 87.94% and from 84.60% to 86.37% when compared to the PFF and ATP methods, respectively.

(Azizi et al., 2022) investigated the scheduling of IoT tasks in a heterogeneous fog network in the research. The main goal of this research was to minimize the system's total energy consumption while fulfilling the task deadlines. If a given task's deadline is not reached, it is assigned to a fog node that delivers the smallest deviation from the task's deadline requirement. Two efficient priority-aware semi-greedy algorithms are developed to fulfill these goals. Extensive experiments are used to assess the effectiveness of the presented algorithms. The results showed that the suggested algorithms outperformed existing algorithms in terms of the proportion of IoT tasks that met their deadline requirement, overall energy consumption and system lifespan, and total amount of deadline violation time.

(Singh & Das, 2022) describe a four-tier cloud-fog-IoMT architecture paradigm based on the dependable MQTT protocol, which allows network scalability at both the edge and fog layers. It implements a dynamic gateway selection mechanism based on the idea of a principal actuator node. Singh's paper proposed the design of a message transfer mechanism using the MQTT protocol for improved medical data delivery, appropriate categorization of medical data using fuzzy logic, and offloading of these classified data using adaptive scheduling combined with dynamic clustering of fog nodes. Load balancing technologies reduce network energy usage and delay. Data offloading was accomplished in three stages: fuzzy based medical data classification, computational capacity and energy-aware fog node clustering, and a storage capacity based adaptive scheduling approach.

(Ghanavati et al., 2022) presented a work scheduling algorithm for fog computing with makespan and energy consumption optimization. The paper demonstrated that the proposed strategy outperforms the baseline approaches significantly.

In the paper by (Tariq et al., 2022), an optimal energy-aware job offloading strategy for the Internet of Vehicles is presented in this research. They proposed offloading that can ensure stability in order to make the Internet of vehicles more dependent. In this research paper, they proposed task offloading as a semi-Markov decision process. The drawback of the proposed strategy based on energy is that it does not take scalability and bandwidth into consideration.

Iftikhar et al., (2023) create a new method known as HunterPlus by adding a Bidirectional Gated Recurrent Unit (GRU) to HUNTER so that it may assess graph inputs both forward and backward. HunterPlus is implemented using an organized methodology that includes training, simulation, assessment, data collecting, model development, and system architecture design. The resource layers, management, and Internet of Things make up the system architecture. The DeFog benchmark is used to collect metrics such as CPU, RAM, disk I/O, response time, energy consumption, and SLA breaches to obtain data for testing and simulation. Furthermore, to improve task scheduling and estimate Quality of Service (QoS) parameters, a Convolutional Neural Network (CNN) model is created and trained. The CNN model generates scheduling decisions based on a matrix that represents the current condition of the cloud-fog environment. Azure virtual machines are used for testing and simulation using the COSCO framework. The result demonstrates that HunterPlus outperforms the state-of-the-art baselines in terms of energy consumption and task completion rates by at least 17% and 10.4%, respectively. In addition, the model's performance variability is smaller than that of other models.

Saif et al. (2023) proposed a new algorithm called Non-dominated Particle Swarm Optimization (NPSO). They are using a mutation operator to broaden the particle population's range and prevent it from entering the local optimal search. At the same time, can enhance the Particle Swarm Optimization (PSO)'s constraint and make it easier to discover the Multiple-objective Problems (MOP) solution. First, they did a quantitative comparison of a set of jobs organized by the FCFS, STML, LLF, and MLLF algorithms in terms of delay. They also do another comparison to find the decrease in all techniques' maximum delays for ten task groups, each group has been assigned with 10 tasks. They found out that in comparison to the other algorithms, MLLF performed best in minimizing the maximum delay by about 11% which a stable decrease in delay when the number of jobs conducted increased. After that, they did a performance comparison of the MOPSO-CD, NSGA-II, and NPSO algorithms with a non-linear optimization method in terms of delay where upper bound of the delay threshold (D_max) is set to 100. It is the MLLF's average delay threshold. There were five groups which are 30, 50, 90, 150, and 200 for the workload. The result shows that NPSO algorithm can use less energy than others algorithm, lowering the delay threshold and helps to lower the transmission latency.

Liu et al. (2023) applied particle swarm optimization (PSO) to get the best computation time and energy consumption in a single fog cluster as well as the best load balance among fog nodes. Then, they are using the time and energy savings from load balancing and created the particle swarm genetic joint optimization artificial bee colony method (PGABC) to optimize work scheduling across fog clusters. In comparison to GABC, ABC, and PSO, the experimental findings demonstrate that the time delay that was computed using the suggested PGABC method in the provided model was decreased by 1.04%, 15.9%, and 28.5%. Not only that, but there was also a 3.9%, 6.6%, and 12.6% reduction in energy consumption, respectively. Due to PSO being able to handle the issue of optimum load balancing for every task in a single fog cluster, a resource-scheduling strategy based on the PSO and the PGABC algorithm (PGABC–PSO) is intended by them also. This strategy has decreased the energy consumption by roughly 9.7%, 33.3%, 32%, and 29.6%, and the delay by approximately 31.25%, 27.8%, 27.8%, and 25.4% when compared to SJF–PSO, PGABC-R, HSF.ABC&PSO, and MFO, respectively.

Singh et al. (2023) provides a resource allocation method using collaborative machine learning (CML) for fog computing enabled by SDN. The resource allocation strategy for the SDN-enabled fog computing environment is linked with the suggested CML model. The outcomes of the suggested technique are tested using the FogBus and iFogSim, utilizing a variety of performance evaluation parameters such execution time, power consumption, latency, and bandwidth utilization. Using the previously described performance evaluation metrics, the resulting findings are compared with other state-of-the-art methods currently in use. According to the data, the suggested method cuts down on 19.35% of processing time, 18.14% of response time, and 25.29% of time delay. Additionally, it saves 21% execution time, 9% network utilization, and 7% energy consumption over the current methods.

Mohammadzadeh et al. (2023) proposed HDSOS-GOA, which is a discrete hybrid version of the SOS and GOA algorithms that randomly runs one of the Symbiotic Organisms Search (SOS) and Grasshopper Optimization Algorithm (GOA) algorithms. It uses learning automata to decide which algorithm to run more frequently. The DVFS-based scientific workflow scheduling problem in the fog computing platform was then resolved using the HDSOS-GOA approach. The workflow's tasks are assigned to the most appropriate virtual machines (VMs) based on the HEFT approach. The optimal DVFS-level VMs are then assigned using the suggested HDSOS-GOA methodology. For the result, they aim to cutting down on scheduling time and minimize workflow scheduling's energy consumption. Four different sizes of scientific procedures are used to execute extensive simulations. The results shown that their methodology beat many optimization methods algorithms, including SPEA2, PSO, SOS, SOA, GWO, ALO, HHO, GOA, PSO-GWO, and GA-WOA algorithm, in terms of energy utilization and the number of VMs employed.

Hajam and Sofi (2023) recommend the enhanced version of the spider monkey optimization (SMO) meta-heuristic algorithm which are semi-greedy task scheduling SMO (sgTS-SMO) and greedy task scheduling SMO (gTS-SMO) for creating effective task scheduling in a fog computing environment. The primary goal is to reduce energy usage and delays while taking deadline and time-violation restrictions into account. The parameters of deadline violation time, makespan, energy consumption, overall cost, and degree of imbalance are used to evaluate the system. According to the data, when gTS-SMO is compared to sgTS-SMO and particle swarm optimization (PSO), it decreases the violation time by 13.86% and 88.38%, respectively. Additionally, the results show that gTS-SMO outperforms in terms of makespan and energy usage. In comparison to sgTS-SMO and PSO, makespan is decreased by 6.28% and 57.75%, while energy consumption is decreased by 5.74% and 53.65%.

Zhao et al. (2024) proposed EOPCO-S algorithm to solve the issue of partial task offloading under known matching between the fog server and terminal device while maintaining optimal energy consumption. Another algorithm called Kuhn–Munkres-based EOPCO-M algorithm is used to solve the best matching issue involving fog servers and terminal devices. The result showed that EOPCO-S can always achieve the lowest energy use when comparing to others scheme. Then, another result proves that when it comes to optimizing task offloading success rate and lowering task processing energy consumption, EOPCO-M performs best compared to other baseline methods such as Hungarian method, random generate and Euclidean distance-based partial task offloading (DPTO).

The metaheuristic algorithm CSO (Cat Swarm Optimization) is employed to manage service activation and resource allocation more effectively Hashemi et al., (2024). A request evaluator receives user requests, sorts them according to priority, and uses the container live migration approach on fog resources to execute them quickly and effectively. By using the container live migration technique, services are moved and positioned more optimally on fog resources and preventing the needless activation of physical resources. To ascertain the initial capacity of physical fog resources, this method makes use of a resource manager to locate and categorize accessible resources. This approach's effectiveness has been evaluated by the application of six metaheuristic algorithms which are Particle Swarm Optimization (PSO), Ant Colony Optimization, Grasshopper Optimization, Genetic, Cuckoo Optimization, and Gray Wolf Optimization in iFogSim simulator. The suggested method outperformed the other six options in terms of various evaluation parameters, including execution time, energy consumption, imbalance, SLA violation, and network life, according to the simulation findings. Based on the result, it is outperformed the other six options in terms of various evaluation parameters, including execution time, energy consumption, imbalance, SLA violation, and network life.

Khan et al. (2024) suggested a novel reactive fault tolerance technique that includes the resubmission of tasks and their execution on processing nodes in the event of a node failure. The monitoring module, which serves as a resource use statistic for submitting, processing, and receiving nodes, resubmits the failed tasks to other executable nodes. The suggested method combined with a modified version of the particle swarm optimization technology reduces energy usage, network bandwidth usage, end-to-end latency, and boosts success and reliability factors. The suggested technique is found to decrease energy consumption by 3%, latency by 5%, network bandwidth uses by 3%, and to boost system reliability by 2% with success rate by 8%. Several trials have been conducted with a maximum of 10 repetitions.

Ghafari and Mansouri (2024) proposes applying nonlinear based chaotic artificial rabbits optimization (NCARO) which is a unique variation of artificial rabbits optimization (ARO), for task scheduling in a fog computing environment (TSNCARO). By utilizing nonlinear and chaotic control settings, the NCARO maximizes ARO. The suggested technique makes advantage of chaotic maps to enhance ARO's exploratory behaviour. The results of the simulation shown that, the suggested TSNCARO algorithm enhanced makespan, service duration, total cost, energy usage, CDER, and PDST in various scenarios. Through the application of a nonlinear mechanism and chaotic maps, NCARO facilitates an easy shift from exploration to exploitation.

Idrees et al. (2024) proposes energy-aware data transmission strategy with decision-making (EDaTAD) that operates on sensor devices and fog gateways, the two level nodes of the fog computing-based TI architecture. At the sensor device level, the EDaTAD applies a Lightweight Redundant Data Removing (LiReDaR) method to reduce the collected data before forwarding it to the fog gateway. A decision-making model is suggested in the fog gateway to help the monitoring team in remote monitoring applications make the right choices. Lastly, it sends the redundant data sets to the cloud for archiving and additional analysis, using a Data Set Redundancy Elimination (DaSeRE) technique. In comparison to the PFF 0.8, PFF 0.75, Bartlett, Tukey, and Fisher methods, respectively, the suggested EDaTAD methodology minimizes the energy consumption from 28.03% up to 74.23%, from 23.55% up to 73.36%, from 14.95% up to 58.86%, from 11.24% up to 61.90%, and from 1.37% up to 58.02%. The findings demonstrate that by eliminating duplicate data reading sets after obtaining them from the sensor devices, the EDaTAD offers superior energy-saving outcomes at the fog gateway compared to the other approaches.

Hossam et al. (2024) deliver an exclusive two-layered hierarchical fog device architecture that maximizes fog node selection for healthcare applications by utilizing cluster aggregation. In order to minimize system latency and lower energy usage in fog computing settings, they provide three effective approaches which are Earliest Deadline First (EDF) Algorithm, Search Nearest Gateway Algorithm and Energy-Aware Module Placement (EAMP) Algorithm. Using the iFogSim toolkit, they thoroughly assess their suggested model and compare it with a cloud-based and latency-aware model. When compared to the Cloud-based approach, the model shows an average reduction in latency of at least 87% and an average reduction in energy usage of at least 76% in four different network topologies. Similarly, the model shows at least a 43% reduction in average latency and 27% reduction in energy use compared to the Latency-aware model.

The Minimal Schedule Time with Energy Constraint (MSTEC) algorithm is a low-delay scheduling algorithm that is suggested for fog computing workflows with energy constraints from Li et al. (2024). Not only that, High Reliability with Energy Constraint (HREC) algorithm also proposed by them to maximize fog computing system stability in mobile circumstances, a workflow with limited energy usage was suggested. The workflow's energy consumption may be efficiently limited by the algorithm, which can also shorten the workflow's completion time. Furthermore, an algorithm known as High Reliability with Energy Constraint (HREC) was put out to optimize the system reliability of fog computing in mobile scenarios, specifically for workflows involving energy constraints. According to our experiments, the HREC algorithm improves system reliability by 22% when compared with the MSTEC algorithm, and the MSTEC algorithm reduces completion times by an average of 16.5% when compared with the baseline algorithm.

Our proposed energy-aware method differs from other works in that we combined two optimization modules which are energy-aware module placement and the dynamic voltage and frequency scaling (DVFS) technique for energy optimization. Energy-aware module placement aims to improve efficiency by placing modules on the fog device which can fulfill its requirements based on the module's estimated minimum energy and MIPS. DVFS reduces cost and enhances resource utilization by adjusting the MIPS of fog devices as close as possible to the MIPS of the module requirement. In fact, by using our proposed energy-aware method, execution time and service delay can also be reduced due to the improvement of overall efficiency.

Offloading Review

Fog computing uses an offloading method that transfers computation power closer to the data source instead of the cloud which is located further from the data source. Implementing computation offloading can help extend the life of the devices' batteries, including laptops, smartphones and network. Fog computing has to be investigated to deal with the issues of work scheduling in cloud data centres at the same time minimizing energy usage. This is because it can decrease the amount of time taken by applications to go live. To achieve this result, fog computing has to use a combination of an IWO (invasive weed optimization) and a cultural evolution algorithm (CEA). In the IWO task scheduling (IWOTS) method, in order to attain high levels of efficiency, employing heterogeneous cluster systems is necessary to organize work. IWOTS makes a point of including both meta-heuristics and heuristic-based algorithms in its algorithms. The IWO technique, which is a meta-heuristic algorithm, is applied to predict the respective dominant of tasks. In addition, the heuristic-based earliest finish time (EFT) method assigns jobs among computer resources to attain the most applicable solution for the job.

There are two offloading destinations: a single server and multiple servers. According to JianYu Wang and his teams (2022), the purpose of offloading is to shift the computation power from a resource-limited mobile device to resource-rich fog nodes so as to enhance the mobile applications' accomplishment. Hence, at the offloading algorithm design phase, selection on cloud servers has to be considered carefully. In order to assure the user experience, by ensuring lower energy consumption and response latency, the runtime workload of an application can be offloaded in sequential execution (to only one server) or in parallel execution (to multiple servers). As the IoT devices are distributed in the network, computation is able to offload to fog servers through WiFi or cellular networks. When a single fog server is not enough to fulfill the latency requirement, then the workload will be shared with other fog servers to be assisted.

Offloading Issues

Task offloading techniques can help IoT devices overcome resource constraints, including computing power, battery life and storage space, which can be particularly useful for computationally intensive workloads on IoT devices, especially mobile devices. As mobile devices are often resource constrained, some of their activities are offloaded to the fog or cloud to improve performance and save battery consumption. We need another object to perform activities instead of IoT devices and deliver the results to them to enable various resource-intensive IoT applications such as augmented reality, virtual reality, facial recognition and multimedia distribution. Task offloading (Wang, B. et. al., 2020) is the term that represents this technology.

Mobile devices, communication connections and fog nodes are the three main components of the work offload process from mobile devices to fog nodes. The mobile device demonstrates how tasks in the IoT program can be disassembled to a smaller size and performed locally or remotely using offloading technologies. The bandwidth, connectivity and mobility of the device determine the quality of data transfer (e.g. WiFi or cellular networks) and offloading technologies are used to move computationally intensive activities from the mobile device to the fog node. This is because the computing power of mobile devices is lower and the computing power of cloud servers is higher than that of fog nodes. Task offloading is responsible for balancing the load of IoT applications at runtime and improving offload performance and system throughput. Load balancing, data management, latency control, security and energy efficiency are a few factors that offload technology can influence in the IoT.

Offloading is a technique for transferring computationally intensive work from resource-limited IoT devices (i.e. offload sources) to resource-rich compute nodes (i.e. offload destinations) to improve the performance of latency-sensitive IoT systems. Methods for task offloading fall into two broad categories, determined by the number of offloading destinations: single and multiple. Single type offloading methods allow offloading of computational tasks to a single fog node for processing in a sequential order, while multiple type offloading methods allow offloading of computational tasks to multiple destinations, e.g. offloading to multiple fog nodes for parallel processing to ensure QoS requirements are met (e.g. lower response latency).

Overview of the Task Offloading Approaches

Ding, Y. et al. (2020) proposed a decentralised computing offload strategy (DCOS) method to build a task segmentation and offloading approach for multi-user multi-mobile-edge scenarios in order to reduce application execution overhead. MAUI was required to divide the application into a series of tasks with correlations, which were represented as a programme call graph. The model solution was then turned into a convex optimization problem, from which the best offloading option was generated.

Lu, H. et al. (2020) proposed a task scheduling algorithm based on MA-DDPG for discrete server selection in order to improve system energy consumption, task latency and task discard rate for mobile and edge devices. It exploited multi-agent continuous learning features to overcome the problem of environmental instability induced by a single decision maker. It combined the SAC algorithm with a maximum entropy reward function to encourage DDPG Actors to do as many actions as possible in order to find more near-optimal path options. It coupled multi-agent DDPG with SAC to tackle the problems of reinforcement learning instability and small learning, as well as offloading energy consumption, latency, and task discarding rate.

Chang, Z. et al. (2020) propose their work to tackle the problem of dynamic task offloading for numerous users while calculating the best power and radio resources. More specifically, the model presented in this research is built on batteries that can absorb energy using energy harvesting techniques and use it to power mobile devices. The authors suggested an approach for dynamically assigning the appropriate power and communication channels using Lyapunov optimization. According to the article, the number of subcarriers influences the cost. The cost falls as the number of subcarriers grows because mobile devices will have more possibilities to discharge their activities. Another finding in the article is that the number of mobile devices has an influence on average cost, which increases as the number of mobile devices grows.

Liu, C. et al (2020) proposed work analyzes a network with a set of end-users and a set of fog nodes. End-user-generated dynamic jobs are independent, yet the CPU cycles required to execute a single bit are the same for all activities. Furthermore, the paper's scenario model includes binary offloading, in which end-users can offload the entire work to an adjacent fog node. If the selected fog node believes that its resources will be insufficient to finish the work by the deadline, the master fog node will offload the task to another selected fog node. As a result, the suggested model implies that each fog node has two queues, one for high priority activities and one for low priority jobs. If a job is transferred from one fog node to another, the suggested model continues to route it to the high-priority queue.

Yang, M. et al. (2021) presented an architecture with a fog layer and a cloud layer, in which vehicles in the fog layer can transfer jobs to the cloud layer. The architecture takes into account both static and mobile fog nodes that can collaborate to offload tasks. It also considers the influence of mobility of task offloading. Thus, the mathematical model portrays coverage as a dynamic variable and enables for the shortest possible task service time while taking storage, bandwidth, and deadline limitations into account. As a result, the authors suggest a task offloading strategy that reschedules work based on their due dates. The suggested approach then selects how to offload tasks depending on capacity and bandwidth restrictions. The extensive simulation findings presented in this research are obtained by employing realistic vehicle trajectories. They demonstrate the suggested policy's performance in terms of task latency, service composition, and task completion rate in various circumstances.

Cheng, Z., et al. (2021) suggested a DRL-based joint deep reinforcement learning (FDRL) architecture to successfully decrease learning loss and privacy leakage during the learning phase. They also suggested a hybrid optimization technique for task offloading and resource allocation approaches based on FDRL. The strategy successfully protects data privacy in the UAV environment, reduces raw data transfer, and reduces learning loss.

Tu et al. (2022) proposed a new approach for dynamic offloading of fog computing that combines long short-term memory (LSTM) with deep learning. Thus, the key contribution of the research is to estimate the load of the fog server in order to optimise the offloading option. Furthermore, the paper's findings suggest that the proposed approach reduces average latency.

Wang, J. et.al. (2022) employed a sequence-to-sequence (S2S) neural network to obtain task dependencies of applications and used non-policy reinforcement learning for task offloading decisions. The S2S neural network was trained based on task DAGs collected periodically. According to the experimental results, the reinforcement learning-based offloading approach achieved better performance than heuristic algorithms, but the application to compute offloading online requires a large amount of training, which causes a large consumption.

Yan, L., et al. (2022) suggested a DRL-based jointly optimal task offloading method that takes into account energy waste when jobs are dropped. The optimization challenges of long-term latency and system energy consumption in task offloading are addressed by integrating a DQN-based reinforcement learning technique with collaborative computing offloading at the cloud edge. However, static simulation tests are used to assess its performance. As a result, in order to evaluate its performance, it must be relocated to a genuine dynamic environment. The drawback of the proposed method based on offloading is it does not take execution time and network usage into consideration.

S. A. Khan et al. (2022) presents a novel approach to task placement that minimizes the cost of employing cloud resources to fulfill the task by executing two-way offloading between cloud and fog. They compare their suggested solution to the current state-of-the-art baseline method and evaluate it on a range of configurations, including big and small data centers with homogeneous and heterogeneous physical machines (PMs) and varying work batches. Their suggested approach actively and dynamically offloaded the cloud's activities while optimizing the use of the fog resources that were available. When the fog can manage the duties, it can return the jobs that are executing on the cloud to it. This significantly reduces both the total cost of employing cloud resources and the time it takes to complete the operation. They assessed their suggested approach for a range of assessing measures and contrasted it with the current cutting-edge baseline technique. In comparison to the baseline approach, the experimental findings show a 40.87% increase in the fog data center's resource utilization and a ×1.68 reduction in the cloud data center's cost.

Reddy and Sudhakar (2023) bringing forth an osmotic-based method for job loading and scheduling. The devices and tasks are classified in the Osmotic Approach (OA)-based heuristic method, and the tasks are allocated to the best devices according to their dynamically available capacity. Using simulated data sets, the suggested scheduling algorithm is compared to more conventional random task loading and round robin task loading procedures. It is discovered that the suggested algorithm performs significantly better than the other algorithms. In terms of certain metrics, the Random Order (RO) and Revised Random Order (RRO) algorithms are significantly outperformed by the Optimal Assignment (OA) algorithm. While OA's execution time is 23.36% to 60.71% faster, its timeliness reliability varies from 94.2% to 100%. Fog device utilization is as high as 97.7%, and throughput improvement varies from 12.43% to 154.58% over RRO and from 57.57% to 199.23% over RO. Furthermore, OA demonstrates a 4.27% to 40.65% turnaround time improvement. These improvements are ascribed to efficient handling of activities with tight deadlines, appropriate task assignment, and effective load balancing.

Task Offloading technique with P4 (TOS-P4) is a new offloading technique that uses Programming Protocol-independent Packet Processors (P4) technology is proposed by Akyıldız et al. (2023). An Intelligent Transportation System (ITS) application scenario is used to assess the suggested scheme, and it is contrasted with a traditional model known as Task Offloading Scheme with Software-Defined Networking controller (TOS-SDN). The testing results show that TOS-P4 is 6.54 times more efficient than TOS-SDN in waiting times for tasks received at Resource Poor (RP) Fog servers when the servers' load status is assessed at 5 s intervals. Additionally, in the TOS-SDN scenario, RP Fog servers have an average task waiting time that is thirty times longer than Resource Rich (RR) Fog servers.

Sulimani et al. (2024) proposes the Hybrid Offloading (HybOff) algorithm, which uses clustering theory to solve problems in both static and dynamic ways, greatly improving load balancing and resource utilization in fog networks. According to experimental data obtained with the iFogSim simulation program, HybOff dramatically lowers offloading messages, distance, and decision-offloading implications. It outperforms static offloading approach (SoA) (64%) and prevalent offloading approach (PoA) (88%), improving load balancing by 97%. Furthermore, it improves system performance 1.6 and 1.4 times more than SoA and PoA, respectively, and raises system utilization by an average of 50%.

Sumona et al., (2024) introduces the ELTO-DQL algorithm, a Deep Q-Learning-based method for task offloading. The algorithm is designed to optimize both user Quality of Experience (QoE) and energy consumption in fog computing environments. The paper emphasizes the balance between reducing service delay and minimizing energy consumption through task offloading to fog nodes, demonstrating improvements in both QoE (by 15%) and energy efficiency (by 19%) compared to existing benchmarks

For workflow applications in FCI with heterogeneous resources and varying communication costs, Shukla & Pandey, (2024) proposed an algorithm called MOTORS algorithm which combine both hybrid optimization-based resource scheduling approach (HORSA) with a fuzzy dominance-based task clustering and overloading technique (FDTCO). The basis of HORSA is the hybridization of HS and GA. The following five workflow datasets have been simulated: Montage, CyberShake, epigenomics, LIGO (inspiral), and SIPHT. Average makespan, average cost, average RUF, and average energy consumption have all been determined. We have compared our suggested MOTORS algorithm to other resource management strategies already in use such as ACO, HPSOGWO, and the MAA algorithm. The task overloading and resource scheduling solution is effectively optimized by the suggested MOTORS algorithm in terms of makespan, cost, resource utilization, and energy consumption. In comparison to ACO, HPSOGWO, and MAA, MOTORS considerably shortens the makespan by 91%, 88%, and 49%, respectively. This variation is acceptable even if the average cost increased by roughly 129%, 112%, and 100% over these methods.

Our proposed computation offloading method can achieve the shortest execution time and the least network usage compared with other work. This is because analysis and testing are carried out on the simulation repeatedly to evaluate the performance of our proposed algorithm. In fact, several simulation parameters are followed during the testing such as the processing capacity of fog devices, RAM of fog devices, network latency, fog device upstream capability, fog device downstream capability, module size and tuple size.

Summary of Related Work

Table 1. Summary Related Work of the QoE

Reference	Problem	Technique	Data	Application
R. Mahmud et al (2020)	Hierarchical, dispersed, and heterogeneous nature of computing instances	fuzzy logic models	IoT devices	iFogSim
Baranwal, G et.al (2020)	High computational complexity of application placement policy	TOPSIS (Technique for Order of Preference by Similarity to Ideal Solution)	IoT devices	IoT system
Nashaat, Ahmed and Rizk (2020)	Reduction of delay in application	Multi-Dimensional QoE (MD-QoE) model	Fog nodes	IoT application
Varshney et al. (2021)	Fog environments having resource heterogeneity, resource limitation and unpredictable nature.	Multi-criteria decision making (MCDM) techniques	Fog computing	Smart application
Zhao et .al (2021)	QoE-driven cross-layer optimization issues	Near-optimal iterative algorithm (EC-VE) and greedy algorithm	IoT devices	IoT system

Saovapakhiran et.al. (2022)	IoT service providers are competing to provide services	QoE-driven architecture	IoT Devices	IoT system
Wang et.al (2022)	Point clouds have large volume of data, difficult to stream in bandwidth-constrained networks	QoE-driven adaptive streaming approach	Point clouds	Point Cloud
Sreenivasu Mirampalli et al., (2022)	Addresses the challenge of efficiently allocating resources in fog-enabled mission-critical IoT applications	Hungarian Maximization Algorithm and fuzzy-based approach for QoE calculations	Parameters for applications (access rate, resource requirements, processing time) and fog instances (round trip time, resource availability, processing speed)	Fog computing networks, particularly for mission-critical IoT applications
Yadav & Baranwal, (2023)	Find an efficient method of selecting task processing positions in IoT edge networks while considering QoE to meet the expectations of different applicants and improve system performance	fuzzy logic models	IoT devices	IoT System
Carvalho & Macedo, (2023)	Degradation of user Quality of Experience (QoE) caused by co-located applications in cloud environments	Deep Learning Models, QoE Estimator Algorithm, Kubernetes Scheduler Extensions, and evaluation on Testbed	QoE metrics	IoT System
Evangeline et al., (2023)	Device Heterogeneity, Limited Bandwidth, Response Time, Resource Allocation, Fault Tolerance, Quality of Experience (QoE), Cost	Guess Fit Algorithm, Cluster Bandwidth Allocation (CBA) Algorithm, Switching Table-based Fault Tolerance Module	Bandwidth in cloud network	IoT devices
Abofathi et al., (2024)	Optimize module placement to meet quality of service requirements, reduce energy consumption, and improve service quality in fog computing environments.	Whale Optimization Algorithm (WOA), Learning Automata, Distributed Learning Automata, Particle Swarm Optimization (PSO), and NSGA-II algorithm.	IoT applications, fog nodes, service placement, energy consumption, delay, and network usage	IoT application
Bikas & Sayıt,	Selection of paths for	Genetic Algorithm-Based	QoE Metrics	Adaptive HTTP

(2024)	Multipath TCP (MPTCP) subflows to maximize the Quality of Experience (QoE) for adaptive HTTP streaming systems.	Path Schelection		Streaming Systems (HAS)
Liu et al., (2024)	Effectively caching video files and managing bitrate adaptation to balance video quality, re-buffering time, and transmission delays	Caching Placement Model, Bitrate Adaptation Model, Video Transmission Model, Utility Function Model	QoE Metrics	Collaborative caching and adaptive bitrate streaming
Islam et al., (2024)	Addressing trade-offs between service latency, availability, QoE, VNF deployment costs.	HPSP - Hyper-heuristic algorithm for NP-hard optimization.	IoT devices	Fog Computing

Work	Observes User Expectations in			Meets Instances Status regarding			Decentralised Management	Prioritised Placement	Deadlin e	Compound QoE Gain
	Service Access	Resource Requirement	Processing Time	Proximity/ Response Rate	Resource Availability	Processing Speed				
H. Santos et al., 2020	✔	✖	✖	✔	✖	✖	✔	✖	✖	✔
M.J. Farooq et al., 2020	✖	✖	✖	✔	✖	✖	✖	✖	✖	✔
A. Tsipis et al., 2020	✔	✖	✖	✔	✖	✖	✔	✖	✖	✔
A. Munusamy et al., 2020	✖	✖	✖	✔	✔	✖	✔	✔	✖	✖
Abd Elaziz et al., 2021	✖	✖	✔	✖	✖	✔	✔	✖	✖	✖
Bichi et al., 2022	✔	✖	✖	✔	✔	✔	✖	✔	✖	✔
Liu et al., 2022	✔	✖	✖	✔	✔	✔	✔	✔	✖	✖
Jasim et al., 2022	✔	✖	✔	✔	✖	✔	✔	✖	✖	✔
M. Sriraghavendra et al., 2022	✖	✖	✖	✔	✖	✖	✔	✖	✔	✖
Guha Roy et al., 2022	✖	✖	✔	✖	✔	✔	✔	✔	✖	✔
Sreenivasu Mirampalli et al., 2022	✔	✔	✔	✖	✔	✔	✔	✔	✖	✔

Yadav & Baranwal, 2023	✔	✔	✔	✔	✖	✔	✔	✔	✖	✔
Carvalho & Macedo, 2023	✔	✔	✔	✔	✔	✔	✔	✔	✖	✔
Evangeline et al., (2023)	✔	✔	✔	✔	✔	✔	✔	✔	✖	✔
Abofathi et al., 2024	✔	✔	✔	✖	✔	✔	✔	✔	✔	✔
Bikas & Sayıt, 2024	✖	✖	✔	✔	✔	✔	✖	✖	✖	✔
Liu et al., (2024)	✔	✔	✔	✔	✔	✔	✔	✔	✖	✔
Islam et al., 2024	✔	✖	✔	✔	✔	✔	✔	✔	✖	✔
Our Work	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔

Table 2.1 Summary Related Work of the Energy aware

Reference	Problem	Technique	Data	Application
Bichi et al. (2022)	High-security environments are harsh and unpredictable, and the technologies used in them must work constantly for an extended period of time.	A sequential application module and a master-worker application module.	Military things data	Energy in IoT devices
Malik et al. (2022)	IoT devices and fog nodes have limited energy, energy-efficient approaches for storage and processing are required in 6G.	Newly developed energy-efficient solutions	Energy-efficient solutions	Energy efficiency in IoT devices / Sensor nodes / Fog nodes
Chuang & Hsiang (2022)	Due to mobile devices keep increasing, hardware resource faced limits, certain applications may not run smoothly.	popularity-aware and energy-efficient offloading mechanism	Upload and return data size	Energy in mobile devices / Fog nodes and offloading
S. Azizi et al (2022)	Green renewable energy with novel dynamic frequency scaling	Energy and performance-aware scheme for the Fog–IoT environment	Blockchain technology	Energy
Nazari Bu-Ali et al. (2022)	Scheduling such dependent jobs in Fog is an NP-hard issue that takes a long time to solve and high energy consumption, leaving it unsuitable for real-time applications.	A multiobjective task scheduling model which includes an intelligent solution named IETIF which combines and leverages the benefits of simulated annealing and NSGA-III algorithms.	Priority tasks in the form of a DAG graph, with graph construction and weighting	IoT devices
Naha et al, (2022)	Due to unique programmes, mobile devices' energy consumption is very dynamic which makes a realistic energy profiling for mobile devices difficult.	energy-aware resource allocation technique with a hybrid approach and a sustainable solution.	Delay, processing time and processing cost status data	Fog nodes, energy-aware

Mordacchini et al. (2022)	It is difficult to maintain highly accessible computer and data infrastructures while minimising system energy usage.	Application placement performing edge-to-edge exchanges (EMC algorithm)	Convergence speed	Energy and Qoe Aware
Feng et al. (2022)	Energy minimization problem is a mixed-integer nonlinear programming that causes more energy consumption.	Novel transmission-strategy-based NOMA transmission in the multi IoT cooperative fog computing system	Computation time and local CPU frequency	Energy in Fog nodes / IoT nodes
Delgado & Famaey (2022)	Capacitors have a limited energy storage capacity, they exhibit intermittent on-off behaviour.	Energy-aware task scheduling algorithm	Voltage behaviour	Batteryless IoT devices
Avgeris et al. (2022)	It is difficult to place computing duties of IoT applications in fog infrastructure.	Fuzzy logic is used to determine the RoE of applications, CCS of instances, and QoE of a user. Then use the Hungarian maximisation assignment technique for mapping.	Resource Gain and Processing Time Reduction Ratio	Energy in IoT devices, QoS and QoE
Sellami et al. (2022)	Problem in establishing appropriate resource allocation and high performance levels while dealing with job management, energy conservation, and ultra-reliable low-latency unpredictability.	Deep Reinforcement Learning is being used to construct job assignment and scheduling methods for SDN-enabled IoT networks.	Computing Intensity status and computing resources on fog nodes	Fog-enabled mobile IoT nodes and QoS, QoE
Idrees et al. (2022)	This massive amount of data results in high communication costs, increased power consumption, and excessive latency at the fog gateway.	Energy-efficient Fog Computing-enabled Data Transmission(EFoCoD) protocol, a Lightweight Data Redundancy Elimination (LiDaRE) Algorithm.	EFoCoD protocol	Energy efficiency in smart sensor nodes
Singh & Das (2022)	Massive amounts of data generated by time-sensitive IoT devices necessitate increased scalability, excessive energy consumption and reduced latency.	Four-tier cloud-fog-IoMT architectural model based on reliable MQTT protocol	MQTT, Client, Broker and Subscriber	Energy, QoS
Ghanavati, Abawajy & Izadi (2022)	Additional resources are required to reduce the conflict between developing IoT applications and resource-constrained IoT devices.	Ant Mating Optimization (AMO) and bi-objective task offloading	Computation intensity status	Energy and Task offloading in IoT devices, QoS
Tariq et al. (2022)	The Internet of Vehicles has become more dependent, so it needs to ensure stability.	Task offloading as semi-Markov decision process (SMDP)	Resource units	Internet of Vehicles, energy-aware and offload
(Iftikhar et al., 2023)	Existing algorithms, including Reinforcement Learning (RL), have drawbacks such delayed adaptation in unstable contexts and poor scalability.	HUNTER include a Bidirectional Gated Recurrent Unit (GRU) (HunterPlus)	Instructions per Second(IPS), RAM, Disk, and Bandwidth consumption	energy used by processors, storage, memory, and network devices
Saif et al. (2023)	Multiple-objective problems (MOP)	Non-dominated Particle Swarm Optimization (NPSO),	Delay threshold	energy consumption within workload groups

Liu et al. (2023)	Inefficient fog devices due to amount of IoT devices	particle swarm genetic joint optimization artificial bee colony algorithm and particle swarm optimization (PGABC–PSO) strategy	Fog node	Energy in fog computing
Singh et al. (2023)	Absence of prerequisites for the majority of contemporary fog computing applications	SDN-enabled fog computing with Collaborative Machine Learning (CML)	performance evaluation parameters	robust resource allocation technique
Mohammadzadeh et al. (2023)	High number of Virtual Machines required for workflow execution	discrete hybrid version of the SOS and GOA algorithms (HDSOS-GOA) algorithms	scientific procedures, number of virtual machines	energy consumption of the scheduling process
Hajam and Sofi (2023)	resource constrained fog nodes for heterogeneous IoT tasks	Greedy task scheduling SMO (gTS-SMO) and semi-greedy task scheduling SMO (sgTS-SMO)	performance evaluation parameters	energy consumption in an active state when fog nodes process tasks
Zhao et al. (2024)	Maximize system energy consumption during task offloading	EOPCO-S and EOPCO-M algorithm	Fog node, IoT devices	Energy consumption during task offloading
(Hashemi et al., 2024)	disordered energy consumption and latency increasement	metaheuristic algorithm Cat Swarm Optimization (CSO) and live migration technique	Fog node	Energy consumption and minimize latency on processing resources.
Khan et al. (2024)	faulty or damaged fog devices result in inaccurate measurements or destruction that will negatively impacts the system's overall performance	fault-tolerant technique using optimization algorithm	data generated by sensors	energy usage as a criterion for any reactive techniques.
Ghafari and Mansouri (2024)	The majority of earlier scheduling plans did not simultaneously take into account the three criteria factors: energy, cost, and service time	Nonlinear and Chaotic version of the ARO algorithm (NCARO) for Nonlinear and Chaotic ARO	private	energy resulting from both active and inactive states
Idrees et al. (2024)	heavy data traffic of IoT applications reality	Energy-aware Data Transmission Approach with Decision-making (EDaTAD)	Actual measurements of detected data from the sensor nodes	energy consumption at sensor devices
Hossam et al. (2024)	Due to resource limitations and device limitations, effectively choosing fog nodes for application modules with different deadline needs and guaranteeing adherence to quality of service (QoS) criteria pose substantial difficulties.	Search Nearest Gateway Algorithm, Earliest Deadline First (EDF) Algorithm, the Energy-Aware Module Placement (EAMP) Algorithm	data sensed by the sensors	energy consumption in fog computing environments
Li et al. (2024)	electricity costs and carbon emissions continue to rise due to the high energy consumption	Minimal Schedule Time with Energy Constraint (MSTEC) algorithm, High Reliability with Energy Constraint (HREC) algorithm	Fog nodes	restricted energy consumption for the workflow

Work Article	**Criteria**						
	Energy Efficiency	Scalability	Latency	Bandwidth	Energy Usage in Fog Nodes	Violation Time and Energy	Delay-aware
Bichi et al., 2022	✖	✖	✔	✔	✖	✖	✖
Malik et al., 2022	✔	✖	✔	✖	✔	✖	✔
Chuang & Hsiang, 2022	✔	✖	✔	✔	✔	✖	✖
Azizi et al, 2022	✔	✖	✔	✔	✔	✔	✔
Nazari Bu-Ali et al., 2022	✔	✖	✔	✖	✔	✖	✔
Naha et al., 2022	✖	✖	✔	✔	✔	✔	✖
Mordacchini et al., 2022	✔	✖	✔	✔	✔	✖	✖
Feng et al., 2022	✔	✖	✔	✖	✔	✔	✔
Delgado & Famaey, 2022	✔	✖	✖	✖	✖	✖	✖
Avgeris et al., 2022	✔	✔	✔	✖	✖	✔	✖
Sellami et al., 2022	✔	✔	✔	✔	✔	✖	✔
Idrees et al., 2022	✔	✖	✔	✖	✖	✖	✖
Singh, N & Das, AK 2022	✔	✔	✔	✔	✔	✖	✔
Ghanavati, Abawajy & Izadi, 2022	✔	✖	✔	✖	✔	✖	✖
Tariq et al., 2022	✔	✖	✔	✖	✔	✔	✔
(Iftikhar et al., 2023)	✔	✔	✖	✖	✔	✔	✔
Saif et al. (2023)	✔	✖	✔	✖	✖	✔	✔
Liu et al. (2023)	✔	✖	✖	✖	✔	✖	✔
Singh et al. (2023)	✔	✖	✔	✔	✖	✖	✖
Mohammadzadeh et al. (2023)	✔	✔	✖	✖	✖	✖	✖
Hajam and Sofi (2023)	✔	✖	✖	✖	✖	✔	✖
Zhao et al. (2024)	✔	✔	✔	✖	✔	✖	✔
(Hashemi et al., 2024)	✔	✖	✔	✔	✔	✖	✖
Khan et al. (2024)	✔	✖	✔	✔	✖	✖	✖

Ghafari and Mansouri (2024)	✔	✖	✖	✔	✖	✖	✖
Idrees et al. (2024)	✔	✖	✔	✖	✖	✖	✖
Hossam et al. (2024)	✔	✖	✔	✔	✔	✖	✖
Li et al. (2024)	✔	✖	✔	✔	✔	✖	✖
Our Work	✔	✔	✔	✔	✔	✔	✔

Table 2.2 Summary Related Work of the Offloading

Reference	Problem	Technique	Performance Parameters		
			Time	Energy	Network
Phan et al., (2021)	Overloaded nodes in task scheduling	Dynamic fog-to-fog offloading in SDN-based fog computing systems	✔	✔	✔
(Kishor & Chakarbarty, 2021)	Difficulty of balancing compute and communication delays while minimizing latency in IoT-Fog situations through task offloading optimization	Smart Ant Colony Optimization (SACO) algorithm	✔	✔	✔
S. A. Khan et al. (2022)	Inadequate for huge workloads requiring a lot of computation on devices with limited resources	Workload/job placement method that performs two-way offloading	✔	✔	✖
Reddy and Sudhakar (2023)	absence of a scheduling algorithm designed to maximize resource utilization at the fog layer, meet job deadlines, and reduce overall execution time	Osmotic Approach (OA)-based heuristic solution	✔	✔	✖
Akyıldız et al. (2023)	Low-quality offloading caused by low bandwidth and high latency	Task Offloading Scheme with P4 (TOS-P4)	✔	✖	✔
Sulimani et al. (2024)	Static offloading (SoA) falls short in heterogeneous networks	Hybrid Offloading (HybOff) algorithm	✔	✔	✔
Shukla & Pandey, (2024)	Workflow applications with diverse resources and varying communication costs in FCI	MOTORS algorithm which combine both hybrid optimization-based resource scheduling approach (HORSA) with a fuzzy dominance-based task clustering and overloading technique (FDTCO)	✖	✔	✖
Our work	Limited resources on fog computing can easily caused overload	Load estimation and optimal task offloading in fog devices	✔	✔	✔

CONCLUSION

In this chapter, much research has been done on QoE placement, energy and offloading. There are a lot of methods that can be used to increase performance in terms of the three criteria. The first and most efficient method for placement is QoE-aware Application Mapping Policy which is proposed in this project. The QoE-aware Application Mapping Policy which includes Fuzzy logic-based approaches and multi-constrained single-objective optimization techniques to improve data processing time and service quality. It also guarantees one-to-one mapping between instances and applications. Besides, this QoE-aware Application Mapping Policy differs from other existing related work because it investigates and studies criteria such as service access rate, required resource amount and responsiveness to data processing problems. The standard designates application placement demands according to user requirements. Their main purpose is to maximize the user's composite QoE gain in terms of criteria such as less crowded networks, capable resource allocation, and shortened operation processing time. Furthermore, this proposed policy is developed in a decentralized manner, therefore, the single point failure issue and management overhead can be avoided.

Moreover, our proposed energy-aware method will be different from other existing related works because the two optimization modules which are energy-aware module placement and the dynamic voltage and frequency scaling (DVFS) technique for energy optimization are combined in this project. Energy-aware module placement aims to improve efficiency by placing a module on the fog device that can fulfill its requirements based on the module's estimated minimum energy and MIPS. DVFS reduces cost and enhances resource utilization by adjusting the MIPS of the fog device as close as possible to the MIPS of the module requirement.

Finally, the offloading method that is proposed by this project is able to achieve the shortest execution time and the least network usage compared with other existing related work. This is because analysis and testing are carried out on the simulation repeatedly to evaluate the performance of our proposed algorithm. In fact, several simulation parameters such as processing capacity of fog devices, RAM of fog devices and network latency, fog device upstream capability, fog device downstream capability, module size, and tuple size are followed during the testing.

Research Methodology and Problem Analysis

In this chapter, the main purpose is to analyse the problem faced and the encountered approaches during the research. The causes and issue of the research topic problems are further illustrated by reviewing plenty of related papers and journals. The research objectives for this research will be done through the methods such as QoE, energy and offloading are explained and introduced in detail. Additionally, a framework is presented for the research methodology. The tools and techniques involved to acquire the data are fully explained.

In this chapter, there will be two main sections which are section 3.1 that describe the approaches to achieve research objectives, problem analysis as well as the faced challenges in this research. Meanwhile, section 3.2 will be the chapter summary of this chapter.

Approaches to Research

The main features of fog computing such as QoE-aware application mapping policy, energy consumption and computation offloading are being studied to further understand the perspective on the development of existing solutions. The research methodology framework is illustrated as shown in Figure 3.1. From Figure 3.1, it can be seen that the analysis will be carried out based on the three main criteria which are QoE-aware mapping, energy and computation offloading based on the collected information from the literature review. The analysis results will assist in understanding and provide a more accurate perspective on the issue that affects the overall performance in fog computing environment. The gathered information will also specify the direction of the research. Besides, the gathered information will also assist in formulating the following objectives in this research.

1. To propose a QoE-aware application mapping policy to improve the satisfaction.

2. To optimise and keep the energy consumption at an optimal level through module placement.
3. To introduce a computation offloading method to prevent overloading on any fog device.
4. To access the suggested solution and measure the work with the existing solutions.

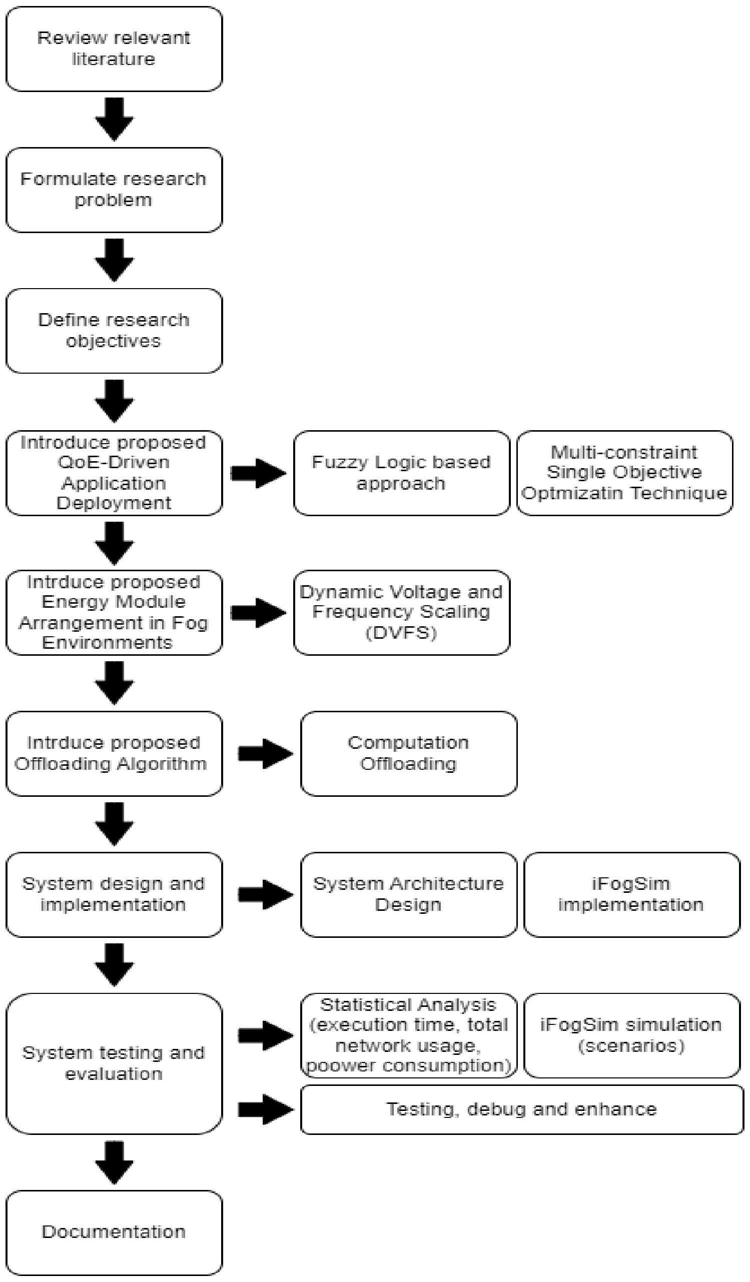

Review relevant literature
Formulate research problem
Define research objectives
Introduce proposed QoE-Driven Application Deployment
Fuzzy Logic based approach
Multi-constraint Single Objective Optmizatin Technique
Intrduce proposed Energy Module Arrangement in Fog Environments
Dynamic Voltage and Frequency Scaling (DVFS)
Intrduce proposed Offloading Algorithm
Computation Offloading
System design and implementation
System Architecture Design
iFogSim implementation
System testing and evaluation
Statistical Analysis (execution time, total network usage, poower consumption)
iFogSim simulation (scenarios)
Testing, debug and enhance
Documentation

Fig. 3.1 Research Methodology Framework

Energy Aware Review

The main features of fog computing such as QoE-aware application mapping policy, energy consumption and computation offloading are being studied to further understand the perspective on the development of existing solutions. The research methodology framework is illustrated as shown in Figure 3.1. From Figure 3.1, it can be seen that the analysis will be carried out based on the three main criteria which are QoE-aware mapping, energy and computation offloading based on the collected information from the literature review. The analysis results will assist in understanding and provide a more accurate perspective on the issue that affects the overall performance in fog computing environment. The gathered information will also specify the direction of the research. Besides, the gathered information will also assist in formulating the following objectives in this research. In this chapter, there will be two main sections which are section 3.1 that describe the approaches to achieve research objectives, problem analysis as well as the faced challenges in this research. Meanwhile, section 3.2 will be the chapter summary of this chapter.

Formulate Research Problem

Challenges of Developing QoE-aware Application Mapping Policy

Different from Cloud data centres, Fog nodes are geographically distributed more closer to the end users. The resource constrained from the fog node will create a significant variance of network round-trip time, data processing speed and resource availability. Therefore, application placement in Fog environments will be a challenging task. There might be different application placement policies required to achieve a certain service level in Fog. The application placement such as Quality of Service (QoS), resource, situation-aware application placement in fog environment are already exploited. However, the impact and effect brought from Quality of Experience (QoE) in Fog-based application placement is still haven't be investigated and researched widely. In some situations, the QoE can be acted as the complement of the QoS. Although separate policy based service is led by QoE and QoS due to there being some subtitles between QoE and QoS.

QoE is accepted to be used for user centric measurement in different service aspects such as to observe user requirements, perception and requirements on a service in different situations. QoE mainly focuses on user interests, therefore, QoE-aware policies can further increase the loyalty of users on a particular service and decrease the service abandon rate. In a fog environment, QoE-aware policies are already used for resource estimation and service coverage optimization. In addition to recovery and service provisioning, the application in Fog Computing is for estimating user QoE which can improve data processing time, resource consumption, and network quality. However, the user interest on different system services might be varied in real-time environments like fog environments. There will also be a frequent change in the QoE domination factors in fog environments. Therefore, to develop efficient QoE-aware policies will be a quite challenging and difficult task.

In IoT, real-time interactions will happen more often than human interventions. Therefore, it is not possible to give feedback after every certain interval to notify QoE. Correspondingly, the significant variety of QoE dominating factors also made prediction-based QoE models to not be successful. There will be difficulties in modifying the placement based on the evaluation of QoE. After placing the application, it is necessary to make modifications based on the evaluation. So, identifying the QoE dominating factors and the combined impact on user QoE will be more viable and prior to application placement. Then, based on the factors, the applications can be placed to the most suitable computing instances so the user QoE will be downgraded. Hence, it is possible to monitor the difference between QoE on specific service and user feedback.

Challenges of Lowering Energy Consumption

Power consumption will be one of the major challenges to be taken into consideration while designing the algorithms and policy. The main purpose for an energy efficient algorithm is to consume the least energy to distribute the tasks and ensure the low latency quality of service at the same time. Available energy budget is another challenge to reduce the energy consumption due to battery-limited power on the devices. For example, IoT devices like sensors are not rechargeable and also have limited energy on the used battery size. Thus, it is required to sleep the sensor while the sensor is unused to save energy by using low-energy communication protocols. Green computing can be achieved by QoS-aware policy due to the non-increment of energy consumption.

Challenges of Distributing Tasks with Computation Offloading in Fog

Computation offloading is a common demand on end devices as well as in fog computing networks. It is very vital to handle fault tolerance and maintain the efficiency of a distributed system in a fog environment due to numerous devices running at the same time. There are certain algorithms that only can be used on closely placed nodes with insignificant delays. In fog environments, multiple fog nodes are used instead of single fog nodes, this because the multiple fog nodes can increase the computing capability to execute the task in order to fulfil low latency requirements. However, the high computation complexity and communication overhead ended up being an obstacle for achieving low-latency and agile response, optimal solutions which are accompanied by global information and centralised control in a fog environment. Thus, the complexity of algorithms is required to be taken into consideration because the simple implementation and operation of algorithms will always ensure the best performance.

Furthermore, there are several aspects that will be required to take into consideration such as the distances between service providing nodes, network links speeds between nodes and the distance between task processed nodes and the client. Thus, it is challenging to design a task distributing algorithm with computation offloading that can work on geographically distributed fog nodes. Also, developing a method to control load balancing mechanisms on the distributed fog nodes that have high delays tolerance in an effective manner is vital. To ensure the efficiency of the performance, the design of load balancing algorithms must be as simple as possible.

Define Research Objectives

There are few objectives to be achieved in this research. First and foremost, the first objective is to propose a QoE-aware application mapping policy to enhance the quality of service with a shorter processing time. Besides, the optimization of energy consumption while assuring the acceptable performance for the distributed task will be the second objective. Furthermore, the third objective is to propose the computation offloading method to prevent overloading on fog devices. Last but not least, the last objective is to evaluate the proposed solutions and use the collected data to compare in terms of efficiency and performance with the existing solutions.

Proposed QoE-aware Application Mapping Policy

To achieve the first objective, several research papers and journals have been reviewed to further understand QoE regards its policy. The reviewed research paper or journal assists in proposing a QoE-aware application mapping policy that can improve quality of service with lesser processing time for data.

QoE-aware application mapping policy uses Fuzzy logic based approaches and multi-constraint single objective optimization technique. The QoE is capable of being influenced by the wide range of user assumptions parameters. Besides, based on various state criteria parameters, the instances of fog computing can also be classified. However, in this research, the access rate, demanded resources and speed limited the user assumption criteria. While, the circulation time, available resource and processing time limited the state criteria.

To develop a QoE-aware policy, it is required to calculate the Degree of Assumption (DoA) and Capacity Class Grade (CCG). To calculate the DoA and CCG, the Fuzzy logic based approach is chosen because the consideration of Fuzzy logic is the best solution for the scalability characteristic in different situations and also the importance of dominance of multiple parameters. According to the assumption criteria and state criteria parameters, the associated Fuzzy sets and rules have been scaled.

After the DoA of application mapping request and CCG of Fog instances which is to get the maximised QoE Gain for mapping of applications are obtained, the multi-constrained single objective optimization technique will be applied. Then, to solve the optimization problem, an optimization solver with a single objective and multiple constraints will be used shortly.

Proposed Energy-aware Module Placement

To achieve the second objective which is to optimise the energy consumption the energy-aware method is proposed. There are two optimization modules proposed which can decrease the total energy consumption, total network usage and execution time. The understanding of energy-aware methods is vital because the performance and capability of infrastructure will be affected.

There are a lot of research papers and journals that show that the capability and performance can be improved by energy optimization. The serious energy wastage is also extremely important. Moreover, the existing proposed algorithms in energy related papers can assist and be a reference on designing an improved energy-aware method.

The first module of energy-aware method is energy-aware module placement which functions to place the incoming task or module to fog devices to fulfil the requirement of the incoming task or module. There are two methods associated in the energy-aware module which include estimating minimum energy of the mobile and MIPS of the module. Comparison between these two methods are made with available fog devices. The comparison will be used to determine whether the fog device is able to handle more incoming tasks and modules in order to place the module. In other words, if the fog device is available to be placed in more modules, the incoming module will be placed in that particular module. Otherwise, the most suitable following fog device will be the next available fog device to place the incoming module.

After the mobile is placed to the fog device, the second module, dynamic voltage and frequency scaling (DVFS) technique will be performed to enhance the energy consumption and resource utilisation. DVFS is used to adjust the MIPS of fog devices in order to fit the MIPS of module requirements based on the MIPS of the module. The purpose of DVFS is to minimise the energy consumption and resource utilisation. In other words, after the fog device is placed by the module, the DVFS is to calculate the new MIPS. Thus, if the incoming module requires less MIPS but the current fog device contains a large amount of MIPS, the DVFS will adjust the MIPS of the fog device to fit the incoming module MIPS.

Proposed Offloading

To achieve the third objective which is to lessen the execution time and decrease the network usage in fog environments, an offloading algorithm is proposed. In the fog environment, there are numerous fog devices in the fog layers that can host applications with one and above instances. It is vital to know how the fog architecture works in terms of the simulation and module mapping function. Besides, it is important to familiarise with the procedure of offloading due to the proposed algorithm being run in simulation and the review will be made during the development process. There will be maximum load of fog devices and the maximum load must be determined at the beginning. The current job load will be added with the new job load when the new job reaches the fog devices. The job will be offloaded to the other fog devices if the result of job load for both current job load and new job load is exceeded on the fog devices. The job will not be executed on the current fog device if the result of job load is exceeded. During the simulation, the offloading analysis and testing will be carried out. During the simulation, there will be few simulation parameters to be followed such as RAM of fog devices, capacity of fog devices, network latency, etc. The result will be generated after the execution is done.

There are other related works such as research papers and journals related to implementation of the offloading method that are also reviewed. The suitable algorithm is chosen and the modification is made based on the chosen algorithm. To achieve the shortest execution time, testing is repeatedly performed. Additionally, the results from the testing simulation are recorded as well as the charts and graphs for further comparison. A set of notation for computation offloading is produced to standardise the coding style. The notation is illustrated in the pseudocode and coding phase of the program. The reader can have a better understanding by referring to the notation set.

To result in better performance in terms of the execution time and network usage during the simulation in the fog environment, an offloading algorithm is proposed. The advantages of this algorithm is it can prevent the fog devices being flooded with every incoming task. Besides, it can determine which fog devices handle the least amount of tasks due to the MIPS. Therefore, all of the fog devices will be fully utilised in the fog environment.

System Design and Implementation

The proposed techniques are transformed into a workable system to achieve all objectives before the proposed techniques can be implemented as a tool to collect data and feedback from the users that are being invited to evaluate the effectiveness of the proposed methods. The evaluation is done through a scenario of virtual reality (VR) game in fog computing environments. There will be two stages in the transformation process which includes system design and the implementation stage.

In the system design stage, the use case diagrams, flowcharts and pseudocodes are used to present all the activities that involve the transformation of the requirements. The implementation of the VR game algorithm involves three stages. By using QoE-aware application mapping policy, energy-aware placement and offloading algorithm to map the module to appropriate mobile devices in order to ensure the optimization of the performance.

During the implementation stage, the simulator called iFogSim is used to implement the proposed methodology. iFogSim is a dynamic environment of IoT applications platform for using to perform simulation. Besides, the tools Eclipse is used as the design and development tool to run iFogSim simulator. The Java programming language is used to write the proposed algorithm. To achieve the predefined objectives, the prototype for every proposed system is well developed and recursively tested.

System Testing and Evaluation

Scenario

Several scenarios have been set to test the proposed solution and solution without energy-aware to achieve the fourth objective. The comparison is made based on a few aspects such as execution time, total power consumption and network usage from the simulation results. Based on the scenario, different aspects will result in different results as shown in Table 3.1. Table 3.1 shows the simulation scenario on the proposed solution without QoE-aware and energy-aware.

Table 3.1 Simulation scenario on fog device and application module arrangement

Scenario	**Fog Device Arrangement**	**Application Module Arrangement**
Scenario 1	Fog Device MIPS increases from end user towards cloud	Module’s MIPS requirements increases from client towards last module
Scenario 2	Fog Device MIPS increases from end user towards cloud	Module’s MIPS requirements decreases from client towards last module
Scenario 3	Fog Device MIPS increases from end user towards cloud	Module’s MIPS requirements is in random order between client and last module
Scenario 4	Fog Device MIPS is in random order between end user and cloud	Module’s MIPS requirements increases from client towards last module
Scenario 5	Fog Device MIPS is in random order between end user and cloud	Module’s MIPS requirements decreases from client towards last module
Scenario 6	Fog Device MIPS is in random order between end user and cloud	Module’s MIPS requirements is in random order between client and last module

Case Study

After the development, the existing solution is used to compare and evaluate the proposed solution. Table 3.2 shows the data gathering techniques for evaluation and comparison. The criteria that is used to compare and evaluate the proposed solution and existing solution includes execution time, energy consumption and network usage. The simulation is completed in the tool iFogSim.

Table 3.2 Data gathering technique used for evaluation and comparison

No.	Title	Data Gathering Technique	Method	Information obtained
1.	Comparison with solution without QoE-aware	Simulation in iFogSim	Quantitative	Comparison result based on energy consumption, execution time and network usage
2.	Comparison with solution without energy optimization	Simulation in iFogSim	Quantitative	Comparison result based on energy consumption, execution time and network usage
3.	Comparison with solution without computation offloading	Simulation in iFogSim	Quantitative	Comparison result based on energy consumption, execution time and network usage
4.	Comparison between solution with QoE-aware	Simulation in iFogSim	Quantitative	Comparison result based on energy consumption, execution time and network usage
5.	Comparison between solution with QoE-aware and energy-aware	Simulation in iFogSim	Quantitative	Comparison result based on execution time, network usage and energy consumption

System Testing and Evaluation

The involved steps and acquired findings will be documented at the end of the research. To show the level of performance and improvement of the development, documentation is necessary to be an evidence to express the findings. Documentation can also express the completeness and accuracy of the work for the reader. The understanding of the reader in terms of the flow of research and the contribution made on existing knowledge is mainly based on the documentation. Besides, documentation will serve as a vital reference in the future work for other researchers.

Summary

In summary, the problem analysis and discussion of methodology used are contained in this research, The problem statement is analysed including the detailed explanation of the cause and issue that lead to the problems. The research methodology framework is developed to achieve the research objectives. The detailed explanation based on the Proposed QoE-aware Application Mapping Policy, Proposed Energy-aware Method, Proposed Computation Offloading Method, System Design and Implementation, System Testing and Evaluation and Documentation are described in the research. The explanation of tools used for the development phase as well as data gathering method is described as well. The following chapter will be presented on the topic of system design and implementation of the proposed solution.

Research Methodology and Problem Analysis

The main topic will be presented in this chapter is the system design and implementation of the proposed solutions. There are two phases in the proposed solution, phase one is QoE-aware application mapping and energy-aware module placement, meanwhile, phase two is the implementation of computation offloading algorithm. The purpose of the proposed solution is to enhance user satisfactions, minimise the execution time, network usage and to optimise the energy consumption. The beginning of this chapter will be the discussion of system architecture design. Then, the system implementation of QoE-aware application mapping, energy-aware module placement and computation offloading will be discussed subsequently. The last section will be the conclusion of this chapter.

System Architecture Design

The information related to the fog computing framework, fog environment simulation and the modelling of fog environment and module placement will be provided in this section.

There are three layers to build up the fog computing architecture which are sensor layer, fog layers and cloud layers. Figure 4.1 shows the fog computing architecture which constitutes layers that are liable for explicit assignments to aid the operation of higher layers.

All service requests are gathered from users in integrated fog and cloud networks. The users are linked with various applications. The users can send service requests to fog nodes via wireless access as well as access to the fog nodes. The fog nodes will respond to the users based on the service request on demand as well. The result will return back to the users through three layers architecture.

The IoT devices linked with applications will perform specific functions based on the request from the end users. Multiple interconnected Applications Modules are divided from Fog-enabled IoT applications. There will be two Applications Modules composed for Fog-enabled IoT applications. The two Application Modules include Client module and Main Application Module. The Client Module will run and process at the user's proximate devices. It will hold the user's preferences and contextual information as well as the deliveries such as acknowledgement or instruction of the Main Application Module. Meanwhile, the Main Application Modules will carry out all the application data operations. The output of the Main Application Module will be regarded as the final product of the Fog-enabled IoT systems.

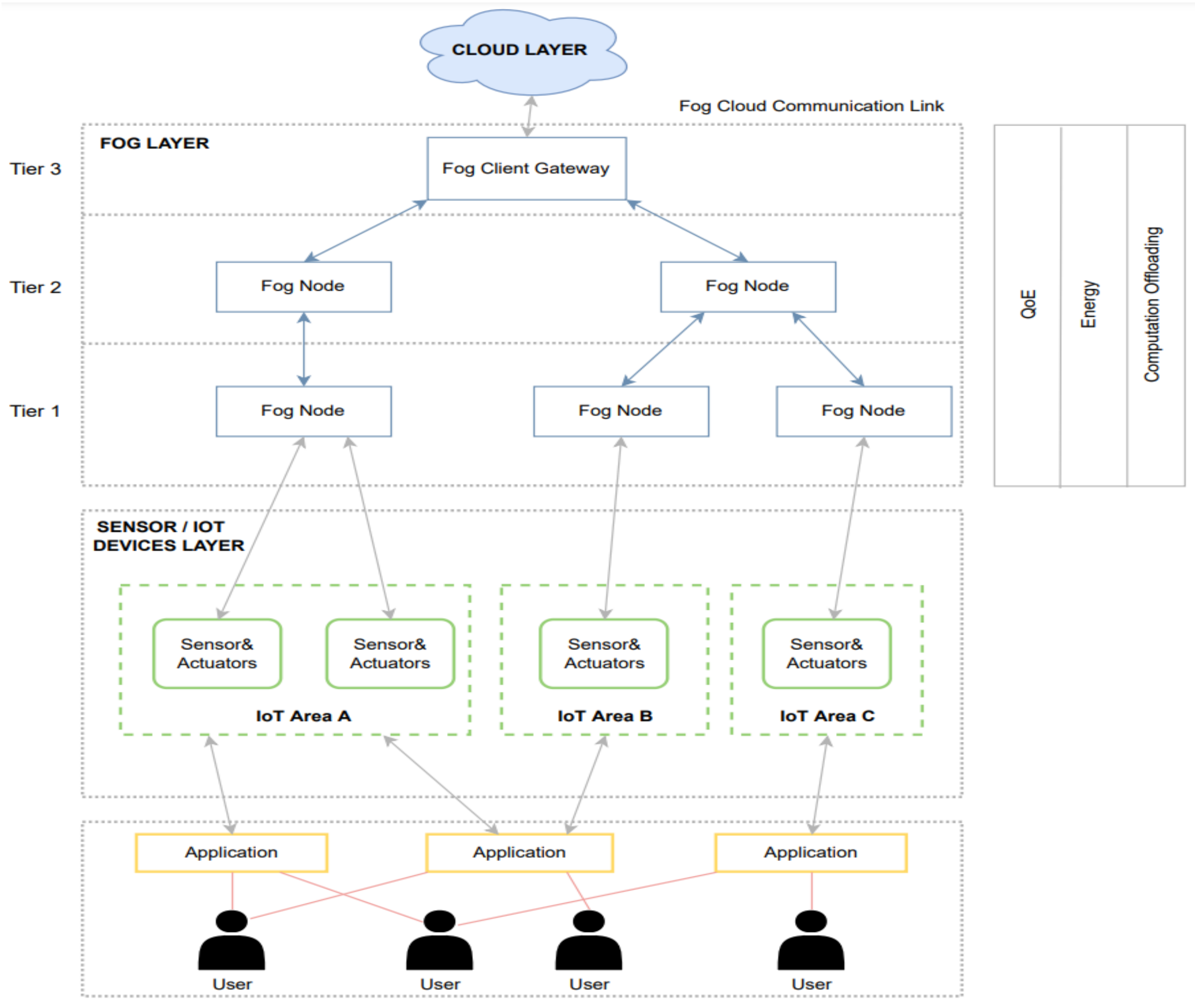


Fig. 4.1 Fog Computing Architecture

The third layer of fog computing architecture will be the sensor or IoT devices layer. The devices in this layer are located the most closest to the end user. This layer is a physical environment which consists of various IoT devices like sensors, actuators, mobile phones, and etc. All of these devices are geographically distributed widely in the world. All of the devices in IoT are modelled by a sensor and actuator and all of these devices are able to emit data. Sensors are used to perform sensing the physical objects or events feature data and transmitting the sensed data for processing and storing to the upper layer. Actuators are used for responding when there are changes in environments imposed by the applications based on the captured information from sensors.

The second layer is a fog layer in which numerous fog nodes are located in this layer. The fog nodes include routers, switches, access points, fog servers, etc. The fog nodes are distributed between the clouds centre and end devices from the end users. The services are obtained by connecting the end devices to the fog nodes. The fog nodes are able to perform computing, transmitting, temporarily storing and receiving sensed data. In the fog layer, the real-time analysis and latency-sensitive application will be accomplished. Additionally, the other side of the fog node is connected to the cloud data centre by IP core network. The interaction and cooperation will be done between fog nodes and cloud data centres for obtaining more powerful computing and storage capabilities.

The cloud layer that performs global or centralised monitoring and control is located at the top layer. The cloud layer consists of multiple storage devices and high-performance servers which are able to perform strong computing and storing capabilities for supporting extensive computation analysis and permanently storage of gigantic amounts of data. Large-scale event detection, long-term pattern recognition and relationship modelling to support dynamic decision making will be the result of cloud scale analytics. One of the major objectives of cloud level analytics is to guarantee the grid and service vendors in order to perform large scale resource and management activities as well as prepare for the blackouts or brownouts.

On top of this, there will be three main aspects to be considered in fog computing. The first aspect is Quality of Experience (QoE). QoE is an acceptable service that is mainly determined based on the users requirements and perception. QoE will provision the service by summarizing the user’s perceptions, requirements and intentions. QoE-aware application mapping is required to guarantee the ensuring quality of experience of the users in terms of enhancing the data processing time and service quality. There are two techniques used for proposing QoE-aware mapping policy. The techniques are fuzzy logic and multi-constraint single objective optimization techniques.

Fuzzy logic is used to perform the calculation of Degree of Assumption (DoA) for application and Capacity Class Grade (CCG) for computing instances. The high combined intensity of associate Assumption Criteria parameters is indicated by high DoA. Whereas, the CCG is to represent the better ability of an instance to fulfil different user assumptions. After performing calculations on both DoA and CCG, the multi-constraint single objective optimization technique is used to maximise the Rating Gain for all application mapping to improve the QoE of the user and also maintain QoS of the user at the same period of time. QoE-aware policy will ensure one to one mapping between applications and instances.

Energy is a second aspect in fog computing. It refers to the total power consumption during fog implementation. Although the increment of quality of service will result in more energy consumed, it is also vital to minimise the power consumption by using an energy-aware module placement. First of all, the fog node will calculate the estimated minimum energy on the incoming module, Then, the estimated minimum energy and MIPS of the incoming module will be compared to the maximum energy and MIPS of the fog device. This process will stop until a fog device is found and able to fulfil the incoming module requirements. Next, the incoming module is placed to the found fog device. The dynamic voltage and frequency scaling (DVFS) that is used to adjust the remaining MIPS.

Lastly, the third aspect is the proposed offloading in the fog layer. The proposed offloading is used to transfer resource intensive computational tasks to another fog device due to the limitation of fog devices such as limited computational power, storage, and energy. To prevent the overall performance being affected, an offloading algorithm is proposed to solve the issues. Although there are numerous fog devices located in the fog layer which are able to host more than one instance for the application, there is the maximum workload for fog devices. When the new task is arrived at the fog device, the system will compare the job load of the task on the current fog device. If the fog device is already occupied by a task, the newly arrived task will be offloaded to the following fog device instead of executing the task at the current fog device.

In short, the three aspects mentioned above are the most vital elements in fog computing environments in order to enhance overall performance for execution of job as well as execution time and usage of network.

The distributed data flow (DDF) model is created for deployment in fog computing and it is the role model of the applications. Applications which contain data processing capabilities are modelled as module collections. Also, based on the data output, it is to produce information which is beneficial to the application. The output data from a processing module for example, module I. This means when the module I is done processing, the result from module I is able to be used as another module such as module J and so on. The data dependency is created between module I and J due to this appearance and it can be performed as an application on directed graph design in this model.

In IoT, the IoT devices use sensors as source of the data. Meanwhile in cloud architecture, the data is known as cloudlet. On the other hand, in fog computing, the data is known as tuples.

Fog Environment Simulation

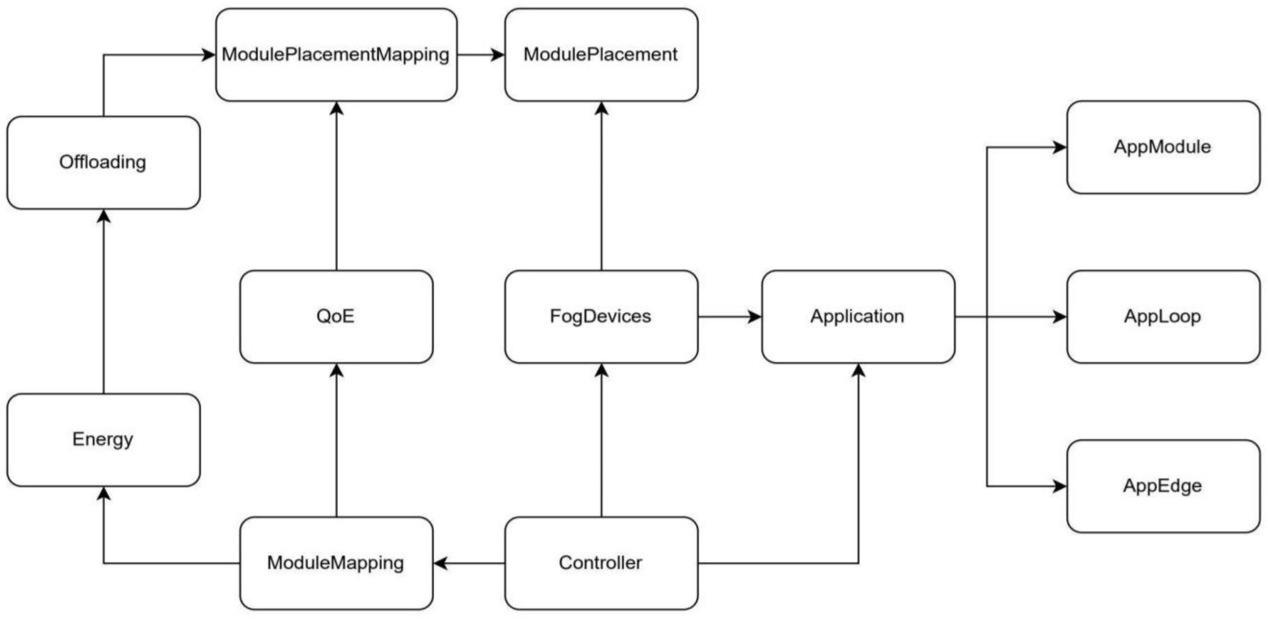


Fig. 4.2 Main classes

Figure 4.2 shows the main classes in fog environment simulation. Fog controller is one of the physical devices that is responsible for building the fog node to deploy the abstraction, it is similar to cluster lead while also granting communication between cloud and fog layers. The Controller is used to control the ModuleMapping, FogDevice as well as the Application.

QoE is used to support module mapping in order to develop an QoE-aware application mapping policy which improves overall user experience and ensures one to one mapping between applications and instances. The second process will be gone through in Energy class which is an energy-aware module placement after application being mapped to fog instance. Energy-aware module placement aims to optimise the energy consumption as well as execution time and usage of the network. Next, the Offloading class is to prevent the overloading of a fog device through the implementation of a proposed offloading algorithm. Through this algorithm, the task will be offloaded to other fog nodes instead of executing when the current fog device is processing a task. After these three processes, the results will be passed to ModulePlacementMapping and finally the module is placed to the suitable fog device by ModulePlacement class. The results will be returned back to application after the task is processed by the fog device. The three classes which are AppModule, AppLoop and AppEdge connect to the application. AppModule serves as the processing elements of fog applications. AppModule will process and send the generated output tuples to next modules in the DAG. AppLoop is an extra class, utilised for determining the loops that are important to the user and controls the process whereas an AppEdge case indicates the information reliance between a couple of application modules and represents a directed edge in the application mode.

Modeling Fog Environment

The target application is a fog computing environment that consists of multiple fog devices which can bring the cloud applications closer to the physical IoT devices at the network edge. Fog device is also known as fog node which is able to process tuples that were sent from other modules hosted on the other fog node hence qualifying fog node as a "mini-cloud" located at the edge of a network that is interconnected by varieties of communication technologies. Virtual machine is the logical data flow presented in a physical fog node in order to fully leverage the processing capability of the fog node. Thus, the virtual machine which is located in the fog devices will process the tuple according to the tuple scheduler. A host is a computer or other device that communicates with other hosts on a network. Hosts on a network include clients and servers that send or receive data, services or applications. Based on research, only one application module is provisioned within a single virtual machine instance to simplify the testing. Figure 4.3 shows the relationship of the related main entities of the proposed solution.

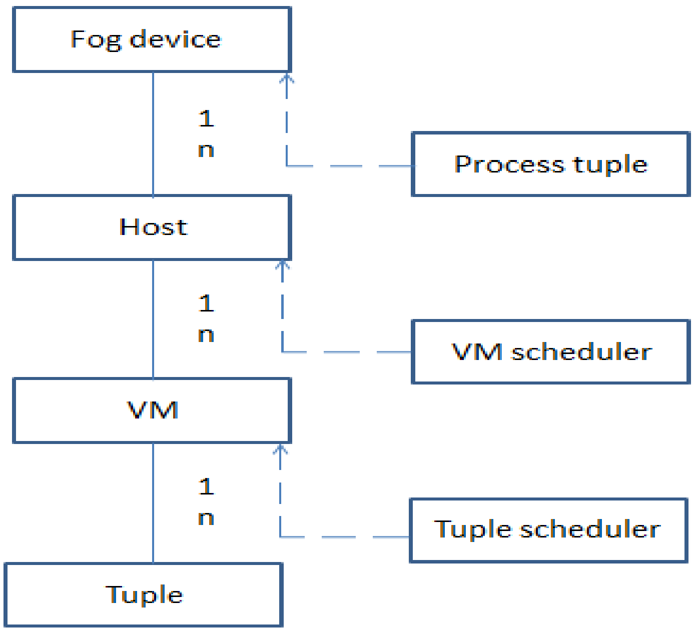


Fig. 4.3 The relationship of the main entities in proposed solution

Application is composed of modules that could be individually hosted on fog nodes to fully leverage the potential of fog devices. The characteristics of modules include maximum MIPS requirement, RAM requirement, Bandwidth requirement, and the tuple frequency. On the other hand, the characteristics of fog devices include MIPS, RAM, bandwidth, link latency, and energy consumption.

iFogSim supports resource management service through two application module placement. "Cloud-only placement" is all modules of an application run in data centres whereas "Edgeward placement" is application modules that are placed close to the edge of the network. However, devices close to the edge of the network may not be powerful enough to host all the applications. The placement policy determines how application modules are placed across Fog devices upon submission of application. The placement process can be driven by objectives such as minimising end-to-end latency, network usage, operational cost, or energy consumption. The class Module Placement is the abstract placement policy that needs to be extended for integrating new policies. The illustration of module placement is shown in Figure 4.4.

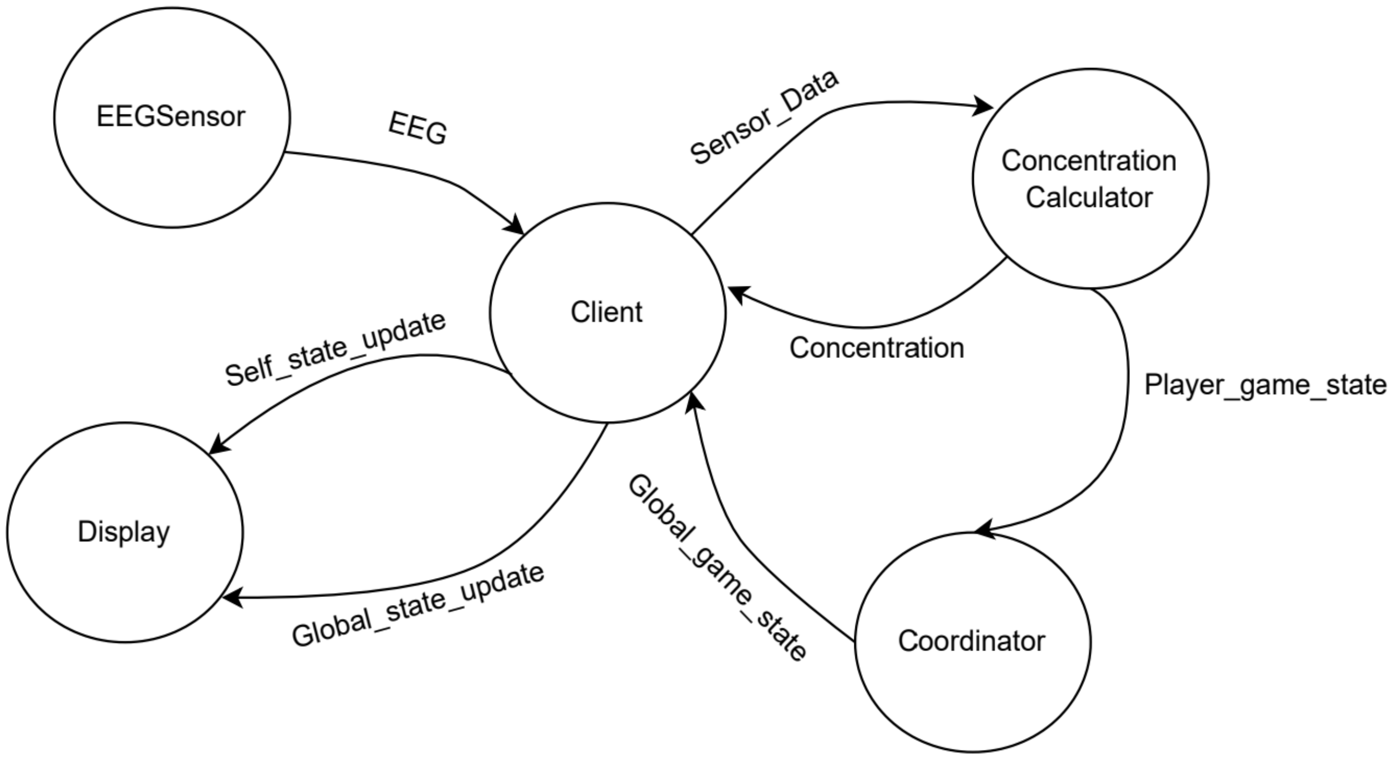


Fig. 4.4 Illustration of module placement

QoE-aware Application Mapping

QoE-aware application will be the first process in phase one of a proposed solution which enhances the user satisfaction.

Architecture of QoE-aware Application Mapping

The computational nodes are equipped with resources such as memory, bandwidth and CPU to run various applications for computational fog nodes (CFN). The resources are virtualized among MSs, micro services, where assignment of applications for execution are conducted in computational nodes. Dynamic provision on the additional resources for a micro service can be conducted from either In CFN, all configured MSs can be operated independently. Controller node is in charge of monitoring and controlling the overall activities of CFN. There is data storage in the controller node that stores metadata that is related to the running application and State Criteria parameters of the MSs. In the controller node, a Capacity Grade Unit is proposed to define a capacity index for each MS based on the State Criteria parameters to ensure that MSs are ranked in accordance with their competence.

Sometimes, the computation of data signals transmitted from IoT devices is facilitated by edge fog nodes, EFNs. For certain Fog-enabled IoT systems, it is assumed that the corresponding EFNs run the Client Module and aid in placing the subsequent module to CFNs in the upper level. In this approach, the connections are established between EFNs and IoT devices. The Client Module is initiated by the Application Initiation Unit of EFNs, through which a user expresses assumptions related to the application to EFNs. EFN services are used to obtain and collect the capacity index of MSs and it is stored in a data storage. Moreover, the data storage keeps user Assumption Criteria and Quantity of Service (QoS) attributes related to the application for further processing. In EFN, there are two individual units which are, Application Mapping Unit and Assumption Degree Unit. For each application mapping request, Assumption Degree Unit calculates a priority value by considering user Assumption Criteria. Other than that, the Application Mapping Unit of EFN carries out mapping of applications to appropriate Fog instances according to the priority value of application mapping requests and the capacity index of MSs respectively. Figure 4.5 shows the architecture for QoE-aware application mapping.

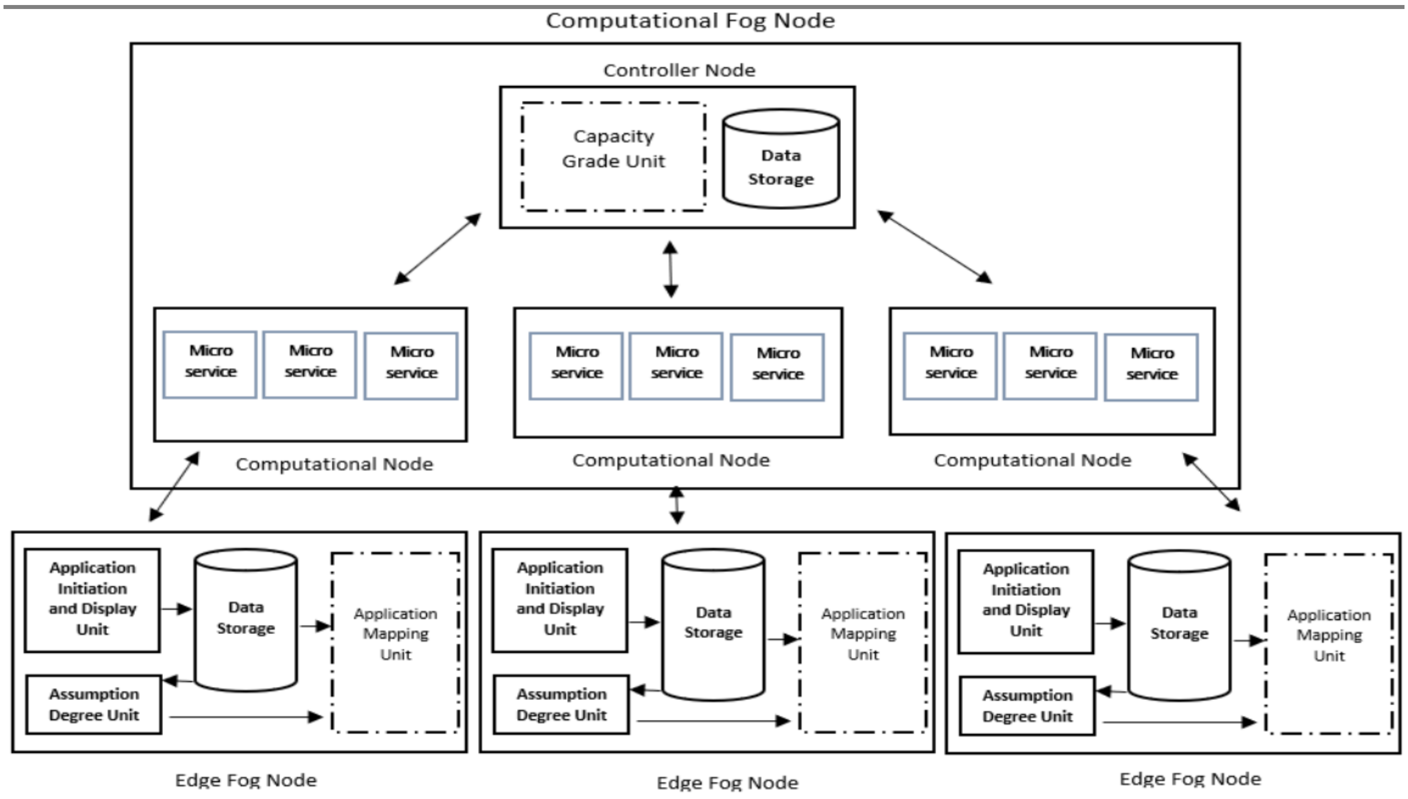


Fig. 4.5 Architecture for QoE-aware application mapping

Flow of QoE-aware Application Mapping

The calculation of a priority value called Degree of Assumption (DoA) will be the essential steps of each application mapping request according to the user assumption parameters, and also to calculate a capacity index called Capacity Class Grade (CCG) of MSs in CFNs in accordance to the state parameters and guarantee the QoE maximised applications mapping to competent MSs using DoA and CCG values. It requires the active participation of Assumption Degree Unit, Application Mapping Units of EFNs and Capacity Grade Unit of CFNs in order to carry out the steps. Figure 4.6 shows the sequence diagram for QoE-aware application mapping.

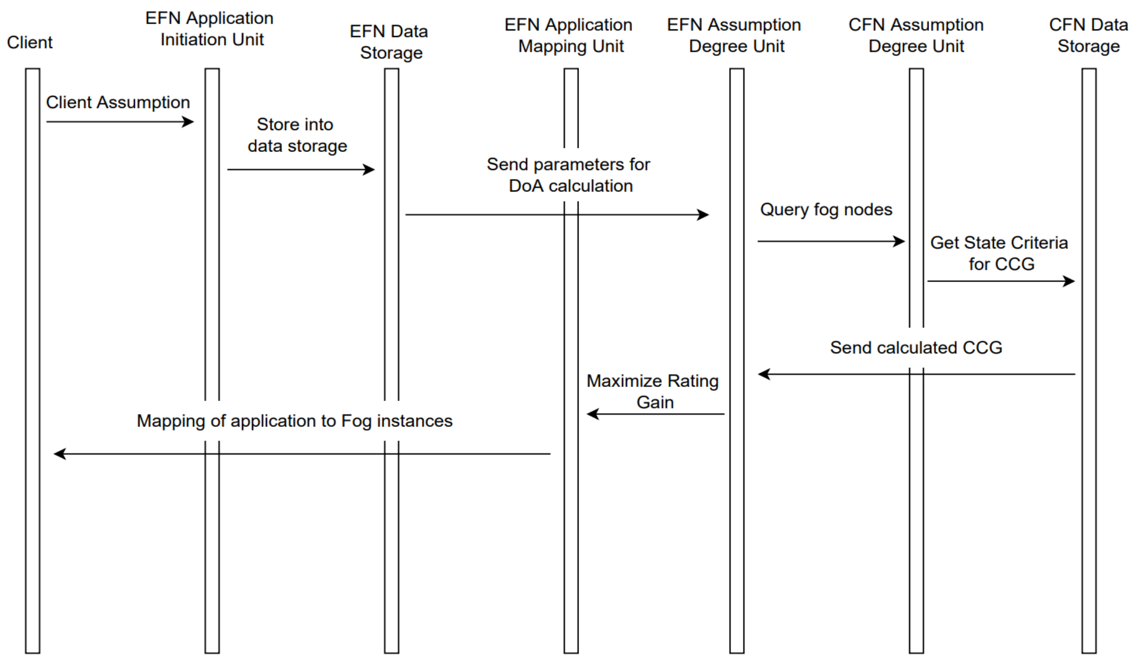

Client
EFN Application Initiation Unit
EFN Data Storage
EFN Application Mapping Unit
EFN Assumption Degree Unit
CFN Assumption Degree Unit
CFN Data Storage
Client Assumption
Store into data storage
Send parameters for DoA calculation
Query fog nodes
Get State Criteria for CCG
Send calculated CCG
Maximize Rating Gain
Mapping of application to Fog instances

Fig. 4.6 Sequence diagram for QoE-aware application mapping

To ensure the best quality of experience for end users, the calculation of two values is vital. The two values are DoA of an application and CCG of a computing instance to effectively map the application to fog instances. There are several steps that need to be followed from the calculation of two values until mapping of application. The first step is to get the Bandwidth, Demanded Resources and Latency Acceptability to store into data storage from the clients assumption. The parameters are sent for DoA calculation through the process of fuzzy inference and defuzzification after assumption parameters are normalised. After calculating DoA, edge fog nodes will query the accessibility of cloud fog nodes about available micro services and CCG values will be associated. Then, State Criteria will be acquired for the calculation of CCG and the CCG is sent once it is calculated. The total Rating Gain of the applications are maximised in the process of mapping applications on that instance. QoE-aware mapping of the applications will be promoted based on the maximum Rating Gain.

Notation and Definition

Table 4.1 shows the QoE-aware application mapping

Symbol	**Definition**
M	Set of all Edge Fog Nodes (EFNs)
N	Set of all Computational Fog Nodes (CFNs)
DoA	Degree of Assumption (DoA) is the priority value that are calculated based on assumption parameters
CCG	Capacity Class Grade (CCG) is a capacity index of micro services according to state parameter to ensure the maximisation of application placement
E_m	Set of all application mapping request in EFN
J_n	Set of all micro services in CFN
$\propto$	Bandwidth parameter in Assumption Criteria
β	Demanded resources parameter in Assumption Criteria
γ	Latency Acceptability parameter in Assumption Criteria
θ	Circulation time parameter in State Criteria
λ	Resource availability parameter in State Criteria
π	Processing speed parameter in State Criteria
A_{e_m}	Assumption Criteria for application $e \in E_m$
S_{j_n}	State Criteria for instances $j \in J_n$
σ_{e_m}	DoA of application $e \in E_m$
Φ_{e_m}	Data signal size for $e \in E_m$
Ω_{j_n}	CCG of instances $j \in J_n$
$K_{\omega}^{e_m}$	Assumption (value) of parameter ω for application $e \in E_m$; $\omega \in \{\alpha, \beta, \gamma\}$
$G_{\varepsilon}^{j_n}$	State (value) of parameter ε for instances $j \in J_n$;$\varepsilon \ \in \{\theta, \lambda, \pi\}$
τ_{ω}	Fuzzy membership function for any A_{e_m} parameter ω
τ'_{ε}	Fuzzy membership function for any S_{j_n} parameter ε
F_a	Fuzzy outcome set for DoA calculation.
F'_c	Fuzzy outcome set for CCG calculation.
$\forall^{f_{e_m}}$	Singleton value for a Fuzzy outcome in (DoA) $f_{e_m} \in \ F_a$ of $e \in E_m$
$\Lambda^{f'_{j_n}}$	Singleton value for Fuzzy outcome in (CCG) $f'_{j_n} \in \ F'_c$ of $j \in J_n$
τ_a	Membership function for any Fuzzy outcome in DoA calculation
τ'_c	Membership function for any Fuzzy outcome in CCG calculation
$v_{j_n}^{e_m} \in \{0,1\}$	Equals to 1 if $e \in E_m$mapped to $j \in J_n$, 0 otherwise
Ar	Normalised access rate
Rr	Normalised resource requirement
Pt	Normalised processing time
ΓBw	Fuzzification bandwidth set

ΓRr	Fuzzification resource requirement set
ΓPt	Fuzzification processing time set
Γi	Fuzzification Inference
Πs	Defuzzification Singleton
s	Singleton
Π	Defuzzification
RoE	Return on Equity Value
CCS	Cloud Computing and Services Value

Calculation of Degree of Assumption (DoA)

Based on the specific fog environment, application placement requests are given distinctive expectation parameters 'range. The amount of the parameter can be chosen unless it didn't reach beyond the range.

Table 4.2 Range of the Parameters for DoA

Parameter/Metrics	**Value in Range**(x_ω, y_ω)
Bandwidth	(3, 18)
Demanded Resources	(4, 16)
Latency Acceptability	(20, 140)

The users end device compromise to the $A_{e_m} \in \left\{K_\alpha^{e_m}, K_\beta^{e_m}, K_\gamma^{e_m}\right\}$regarding an application e_m to the system through the Application Initiation Unit. The data storage will contain the A_{e_m} and it is sent to the Assumption Degree unit of EFN m. A_{e_m} which contain three parameters and the range. The units of the values vary. The values of each parameter are normalised to simplify further calculation. The result of the normalisation will fall in between -1 and 1 by using Eq.4.1:

$$\underline{K_\omega^{e_m}} = 2\left(\frac{K_\omega^{e_m} - x_\omega}{y_\omega - x_\omega}\right) - 1 \qquad (4.1)$$

$K_\omega^{e_m}$ is the normalised value for criteria ω within the range $[x_\omega, y_\omega]$. Each criteria in $[x_\omega, y_\omega]$, is defined based on the scope for every criteria of Table 4.2 offered in the Fog Environment. In the other words, x_ω refer to the minimum value of the range of parameters, y_ω refers to maximum value of the range of parameters In Assumption Degree Unit, a Fuzzy logic based approach is used to calculate the $\sigma_{e_m}$of each application from the normalised parameter in A_{e_m}.

Fuzzification Module for DoA Calculation

Fuzzification is used to convert the crisp input values into fuzzy values by using the information in the knowledge base. In fuzzification, the crisp inputs which are x and y are taken to determine the degree whether they belong to which of the appropriate fuzzy sets. The standardised value $\underline{K_\omega^{e_m}}$ of any A_{e_m} parameter ω is transformed into an equivalent fuzzy dimension through associate membership function τ_ω. This work involved membership functions of different Assumption Criteria from three different fuzzy sets. The following are the fuzzy sets:

- Bandwidth: $Bw \in \{Extremely\ Low, Low, Medium, High, Extremely\ High\}$
- Demanded resources: $Dr \in \{Extremely\ Small, Small, Medium, Large, Extremely\ Large\}$
- Latency Acceptability: $La \in \{Extremely\ Slow, Slow, Moderate, Fast, Extremely\ Fast\}$

In this case, the following $K_\omega^{e_m}$ value will be given as 6 per seconds, 7 CPU cores and 110 ms to Bandwidth, Demanded Resources and Latency Acceptability respectively, as shown in Table 4.3.

Table 4.3 Value of DoA Calculation

Parameter	Bandwidth (per seconds)	Demanded Resource (CPU cores)	Latency Acceptability (ms)
x_ω	3	4	20
y_ω	18	16	115
$K_\omega^{e_m}$	6	7	100

After the value is prepared, it takes fuzzy input and applies it to the antecedents of fuzzy rules, then it can start to calculate the Degree of Assumption (DoA) as shown in Table 4.4 based on Eq.4.1:

Table 4.4: Result of Calculation of DoA

Bandwidth	Demanded Resource	Latency Acceptability
$\underline{K_{\propto}^1} = 2\left(\frac{6-3}{18-3}\right)-1$ $\underline{K_{\propto}^1}$ = -0.60	$\underline{K_{\beta}^1} = 2\left(\frac{7-4}{16-4}\right)-1$ $\underline{K_{\beta}^1}$ = -0.50	$\underline{K_{\gamma}^1} = 2\left(\frac{100-20}{115-20}\right)-1$ $\underline{K_{\gamma}^1}$ = 0.68

The membership degree, $\tau_\omega\left(\underline{K_\omega^{e_m}}\right)$ for any normalised value for criteria ω based on respective fuzzy sets in Figure 4.7. Table 4.5 is fuzzy sets after assumption parameters that are normalised. The fuzzy set will be arranged in:

- ❖ Bandwidth:
  - ➢ $\tau_\omega\left(\underline{K_{\propto}^1}\right) \rightarrow Bw\colon\{\ Extremely\ Low, Low, Medium, High, Extremely\ High\}$
- ❖ Demanded Resource:
  - ➢ $\tau_\omega\left(\underline{K_{\beta}^1}\right) \rightarrow Dr\colon\{Extremely\ Small, Small, Medium, Large, Extremely\ Large\}$
- ❖ Latency Acceptability:
  - ➢ $\tau_\omega\left(\underline{K_{\gamma}^1}\right) \rightarrow La\colon\{Extremely\ Slow, Slow, Moderate, Fast, Extremely\ Fast\}$

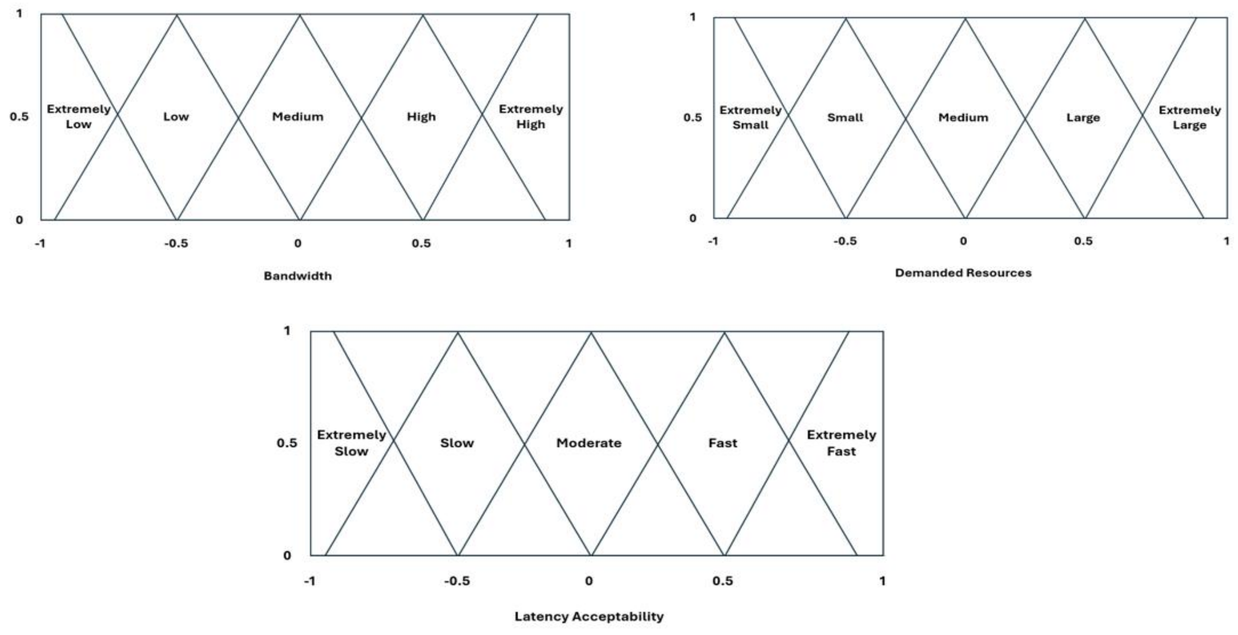


Figure 4.7: Membership function of assumption criteria parameters

Before the normalisation process, it is assumed that the value of parameter $\propto$ in application. Fuzzy sets can be displayed in many shapes. However, triangles or trapezoids can usually fully express expert knowledge and can greatly simplify the calculation process. Figure 4.8 shows how Fuzzy logic separates the area for Bandwidth fuzzy sets.

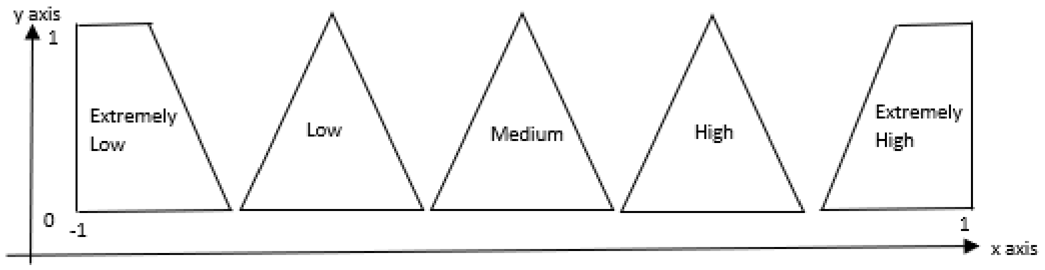


Fig. 4.8 Area separation for Bandwidth fuzzy sets

After the normalisation process is complete in Table 4.4, the answer for $\underline{K_{\propto}^{1}}$ is = -0.60. $\underline{K_{\propto}^{1}}$ refer to the normalised application 1 under Bandwidth parameter $\propto$. Based on the result, it is shown that -0.60 of the x axis hit 0.25 and 0.75 on the y axis of its respective membership set in the Extremely Low and Low area. Refer Figure 4.9, therefore it is concluded that the fuzzy set for $\tau_{\omega}\left(\underline{K_{\propto}^{1}}\right) \rightarrow Bw\colon\{0.25,0.75,0.0,0.0,0.0\}$.

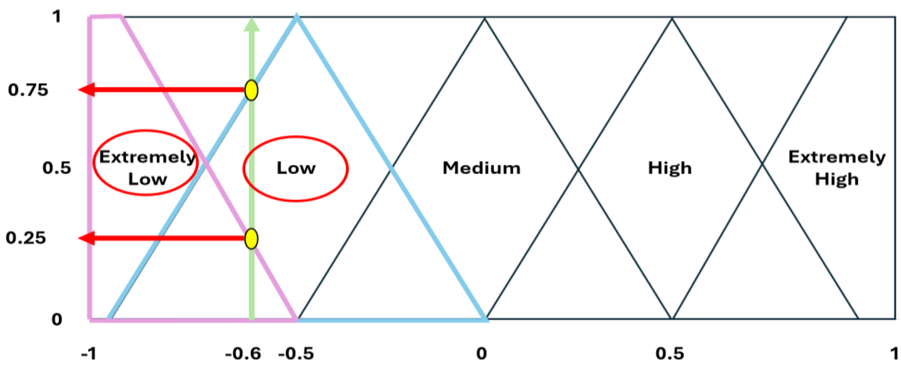


Fig. 4.9 Bandwidth normalised value intercepted Extremely Low and Low area

In parameter of Demanded Resources, $\underline{K_{\beta}^{1}}$ is = -0.50. $\underline{K_{\beta}^{1}}$ refer to the normalised application 1 under Demanded Resources parameter β. Based on the result, it is shown that -0.50 of x axis hit 0.00 and 0.00 on y axis in Extremely Small and Small area on y-axis of its respective membership set correspondingly, therefore it is concluded that the fuzzy set for $\tau_{\omega}\left(\underline{K_{\beta}^{1}}\right) \rightarrow Dr\colon\{0.0,0.0,0.0,0.0,0.0\}$.

In parameter of Latency Acceptable, $\underline{K_{\gamma}^{1}}$ is = 0.68. $\underline{K_{\gamma}^{1}}$ refer to the normalised application 1 under Latency Acceptability parameter γ. Based on the result, it is shown that 0.50 of x axis hit 0.50 on y axis in Fast and Extremely Fast membership sets respectively, therefore it is concluded that the fuzzy set for $\tau_{\omega}\left(\underline{K_{\gamma}^{1}}\right) \rightarrow La\colon\{0.0,0.0,0.0,0.50,0.50\}$. Table 4.5 shows the examples of fuzzy sets after assumption parameters being normalised.

Table 4.5: Fuzzy sets after assumption parameters being normalised

Parameter	**DoA (K_{ω}^{em})**	**Intercepted Area of Y-axis (Normalised Value)**					$\tau_{\omega}\left(\underline{K_{\omega}^{em}}\right)$
Bw	-0.60	Extremely Low (0.25)	Low (0.75)	Medium (0.00)	High (0.00)	Extremely High (0.00)	{0.25,0.75,0.00, 0.00,0.00}

Dr	-0.50	Extemely Small (0.00)	Small (1.00)	Medium (0.00)	Large (0.00)	Extremely Large (0.00)	{0.00,0.00,0.00, 0.00,0.00}
La	0.68	Extremely Slow (0.00)	Slow (0.00)	Moderate (0.00)	Fast (0.52)	Extremely Fast (0.48)	{0.00,0.00,0.00, 0.52,0.48}

Fuzzy Inference Module for DoA Calculation

Fuzzy inference is used to formulate a mapping from a given input to an output using fuzzy logic. Then, the mapping will provide the basis from which decisions can be made or patterns can be identified. During fuzzy inference, corresponding fuzzy outputs are determined by mutually comparing fuzzy inputs with the help of fuzzy rules. The fuzzy rules are set in such a way that approximately stringent assumption parameters like large resource demand are given higher weight. As a result, the DoA value for the requests will be more aligned with the stringent assumption parameters compared to flexible parameters like moderate latency acceptable and medium bandwidth. After that, the system needs to be tuned and evaluated to see if the fuzzy system meets the requirements specified at the beginning. The surfaces can be generated by using the fuzzy logic toolbox to analyse the performance of the system. The fuzzy rules used to calculate DoA are shown in Figure 4.10 while the results of fuzzy inputs (Assumption Criteria parameters) comparison based on the fuzzy rules are shown in Appendices.

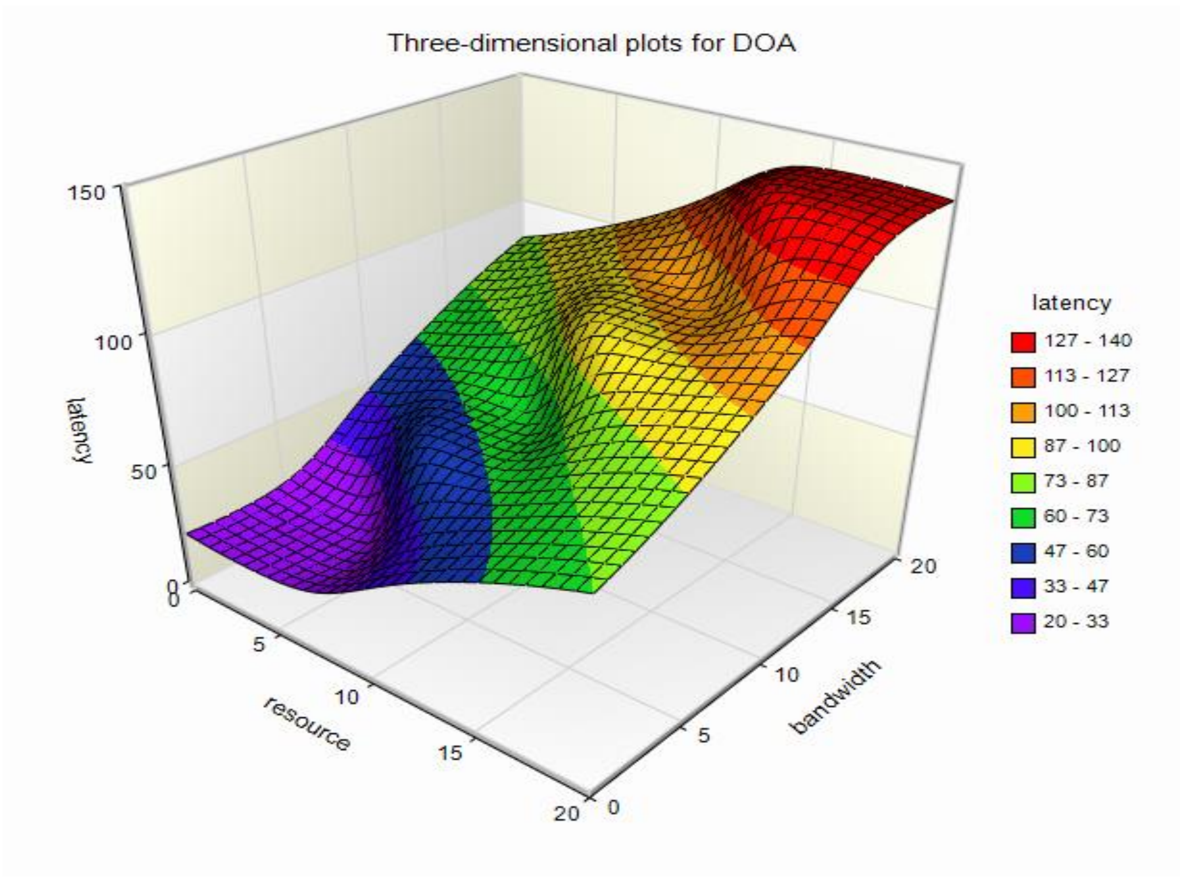
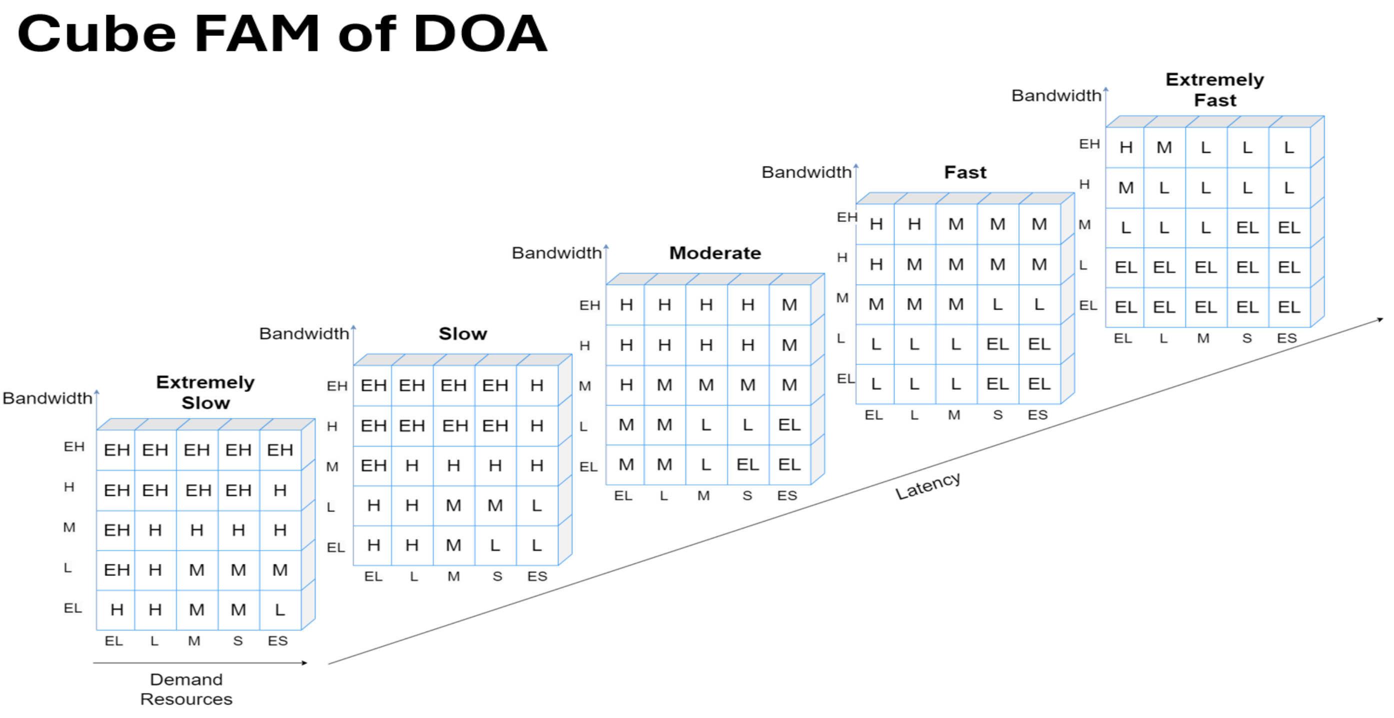


Fig. 4.10 Fuzzy rules for DoA calculation

The membership degree for each of the fuzzy outputs refers to the highest membership degree of each compared parameter from the corresponding fuzzy set. The figure 4.10 is based on table 4.2 Range of the Parameters for DoA. The equation used to determine the membership degree is shown in Eq.4.2:

$$\tau_a\left(f_{e_m}\right) = \left(\tau_\alpha\left(\underline{K_\alpha^{e_m}}\right), \tau_\beta\left(\underline{K_\beta^{e_m}}\right), \tau_\gamma\left(\underline{K_\gamma^{e_m}}\right)\right) \qquad (4.2)$$



Extremely Slow

Bandwidth	EL	L	M	S	ES
EH	EH	EH	EH	EH	EH
H	EH	EH	EH	EH	H
M	EH	H	H	H	H
L	EH	H	M	M	M
EL	H	H	M	M	L

Slow

Bandwidth	EL	L	M	S	ES
EH	EH	EH	EH	EH	H
H	EH	EH	EH	EH	H
M	EH	H	H	H	H
L	H	H	M	M	L
EL	H	H	M	L	L

Moderate

Bandwidth	EL	L	M	S	ES
EH	H	H	H	H	M
H	H	H	H	H	M
M	H	M	M	M	M
L	M	M	L	L	EL
EL	M	M	L	EL	EL

Fast

Bandwidth	EL	L	M	S	ES
EH	H	H	M	M	M
H	H	M	M	M	M
M	M	M	M	L	L
L	L	L	L	EL	EL
EL	L	L	L	EL	EL

Extremely Fast

Bandwidth	EL	L	M	S	ES
EH	H	M	L	L	L
H	M	L	L	L	L
M	L	L	L	EL	EL
L	EL	EL	EL	EL	EL
EL	EL	EL	EL	EL	EL

Fig. 4.11 Fuzzy rules for DoA calculation

From Figure 4.11, the bandwidth will be divided into 5 types which are 'EH', 'H', 'M', 'L', 'EL' and represent 'Extremely High', 'High', 'Medium', 'Low' and 'Extremely Low'. The resource demand is divided into 5 types which are 'EL', 'L', 'M', 'S', 'ES' and represent 'Extremely Large', 'Large', 'Medium', 'Small' , 'Extremely Small'. Using the comparison of the first fuzzy set as an example, with the Extremely Low Bandwidth, Extremely Large Demanded Resource and Extremely Slow Latency Acceptance, the fuzzy output will be on an High level. The illustrative explanation is shown in Figure 4.12. Any u number of fuzzy rules can be triggered based on the Assumption Criteria parameters.

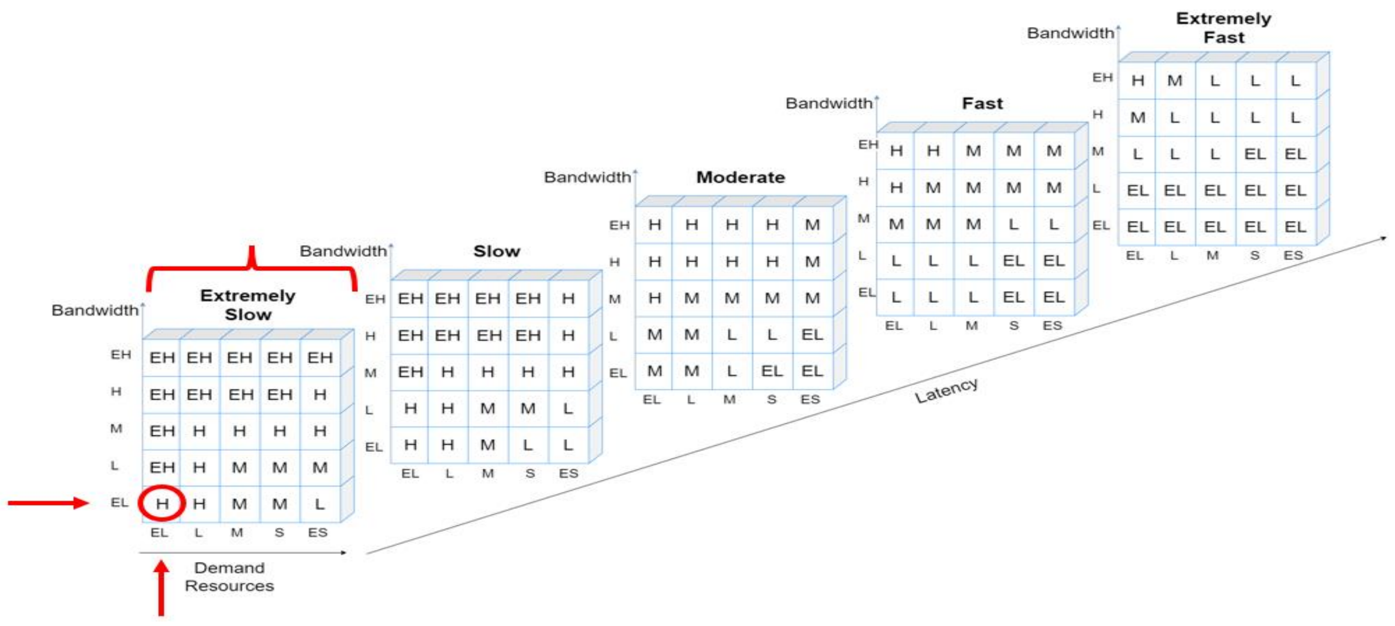


Fig. 4.12 Illustrative explanation on getting fuzzy output for DoA calculation

Defuzzification Module of DoA calculation

The maximum rating of the application for that fuzzy output is represented by a value called singleton value which is set in a way that could make the logical difference on having fuzzy outputs obviously visible. In this case, the singleton value for fuzzy output is set as 'Extremely High', 'High', 'Medium', 'Low' and 'Extremely Low' as 10, 8, 6, 4 and 2 respectively. Fuzziness helps us evaluate the rules, but the final output of the fuzzy system must be a clear number. The input of the defuzzification process is the aggregate output fuzzy set, and the output is a single number. For defuzzification, Fuzzy logic is applied on different parameters of the Assumption Criteria to obtain the exact DoA for application. The equation used to obtain DoA is shown in Eq.4.3:

$$\sigma_{e_m} = \frac{\sum_{z=1}^{z=u} \tau_a(f_{e_m}^z) \times \forall_z^{f_{e_m}}}{\sum_{z=1}^{z=u} \tau_a(f_{e_m}^z)} \qquad (4.3)$$

Each membership degree will be multiplied with the corresponding singleton value, then all the values resulted from the multiplication will be summed up and lastly, be divided by the sum of all membership degrees. σ_(e_m) refers to the exact DoA obtained for application e_m. The Application Mapping Unit will then use σ_(e_m) to map the application to a suitable Fog computing instance.

Calculation of Degree of Assumption (CCG)

Table 4.6: Range of the Parameters for CCG

Parameter/Metrics	**Value in Range** $(x'_\varepsilon, y'_\varepsilon)$
Circulation Time	(2, 17)
Available Resources	(3, 15)
Processing Time	(40, 160)

CCG is calculated after the calculation of DoA has been done for each application placement request. At this stage, EFN m will question the accessibility of CFN n on the available micro services j_n and associate CCG values. For every micro services in a CFN, the CCG is calculated in Capacity Class Grading unit from the correlated State Criteria, S_(j_(n))∈ {G_θ^(j_n),G_λ^(j_n),G_π^(j_n) }. The calculation steps are similar to calculation for Degree of Assumption (DoA). The values for State Criteria parameters are different, therefore the normalisation process is carried out using Eq.4.4:

$$\underline{G_\varepsilon^{j_n}} = 2\left(\frac{G_\varepsilon^{j_n} - x'_\varepsilon}{y'_\varepsilon - x'_\varepsilon}\right) - 1 \qquad (4.4)$$

$G_\varepsilon^{j_n}$ is the result after the normalisation process for criteria ε within the range $[x_\varepsilon, y_\varepsilon]$. The range is set based on the capacity of the Fog environment for those respective parameters. After normalisation, a Fuzzy logic based approach is used for further calculation.

Fuzzification Module for CCG Calculation

The membership τ'_ε is use to determine the membership degree of normalised $G_\varepsilon^{j_n}$ value to relate fuzzy sets. This fuzzy sets for each parameter are listed as follow:

i. Circulation time: $Ct \in \{Extremely\ Short, Short, Average, Long, Extremely\ Long\}$

ii. Resource Availability: $Ra \in \{Critical, Limited, Adequate, Sufficient, Plentiful\}$

iii. Processing Time: $Pt \in \{Minimal, Fair, Acceptable, Efficient, Optimal\}$

In this case, the following $G_\varepsilon^{j_n}$ value will be given a sample as 11 per seconds, 12 CPU cores and 85 ms to Circulation Time, Resources Availability and Processing Time respectively, as shown in Table 4.7.

Table 4.7 Value of CCG Calculation

Parameter	Circulation Time (per seconds)	Resource Availability (CPU cores)	Processing Time (ms)
x'_ε	2	3	40
y'_ε	17	15	160
$G_\varepsilon^{j_n}$	11	12	85

After the value is prepared, calculate the Capacity Class Grade (CCG) as shown in Table 4.8 based on Eq.4.4:

Table 4.8: Result of Calculation of CCG

Circulation Time	Resources Availability	Processing Time
$\underline{G_\theta^{j_n}} = 2\left(\frac{11-2}{17-2}\right)-1$ $\underline{G_\theta^{j_n}}$= 0.20	$\underline{G_\lambda^{j_n}} = 2\left(\frac{12-3}{15-3}\right)-1$ $\underline{G_\lambda^{j_n}}$= 0.5	$\underline{G_\pi^{j_n}} = 2\left(\frac{85-40}{160-40}\right)-1$ $\underline{G_\pi^{j_n}}$= -0.25

The membership function, $\tau_\omega\left(\underline{G_\varepsilon^{j_n}}\right)$ for state criteria parameters is shown in Figure 4.13, while examples of fuzzy sets after state parameters that are normalised are shown in Table 4.9. The fuzzy set will be arranged in:

- Bandwidth:
  - $\tau_\omega\left(\underline{G_\theta^{j_n}}\right) \rightarrow Ct\!: \{Extremely\ Short, Short, Average, Long, Extremely\ Long\}$
- Demanded Resources:
  - $\tau_\omega\left(\underline{G_\lambda^{j_n}}\right) \rightarrow Ra\!: \{Critical, Limited, Adequate, Sufficient, Plentiful\}$
- Latency Acceptability:
  - $\tau_\omega\left(\underline{G_\pi^{j_n}}\right) \rightarrow Pt\!: \{Minimal, Fair, Acceptable, Efficient, Optimal\}$

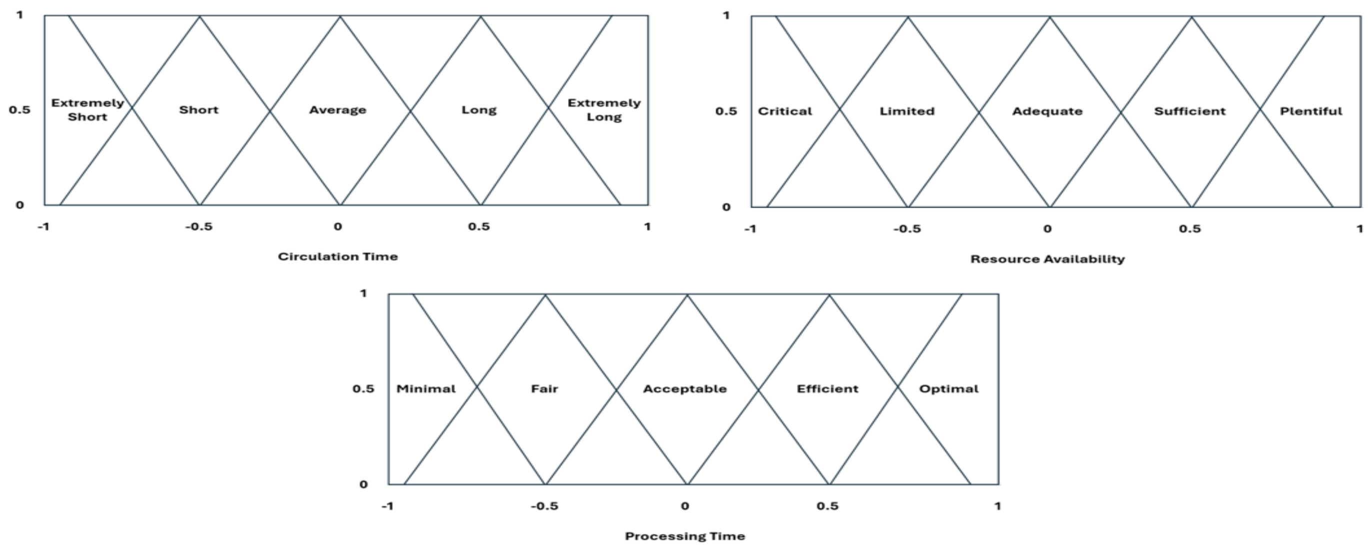


Fig. 4.13: Membership function of state criteria parameters

After the normalisation process is complete in Table 4.8, the answer for Circulation Time, $\underline{G_{\theta}^{j_n}}$ *is = 0.20.* $\underline{G_{\theta}^{j_n}}$ *refer to the normalised application 1 under Bandwidth parameter* θ*. Based on the result, it is shown that 0.20 of x axis hit 0.5 on y axis of its respective membership set in Average and Long area, therefore it is concluded that the fuzzy set for* $\tau_{\omega}\left(\underline{G_{\theta}^{j_n}}\right) \rightarrow Ct\colon\{0.0,0.0,0.55,0.45,0.00\}$.

In parameter of Resource Acceptability, $\underline{G_{\lambda}^{j_n}}$ *is = 0.50.* $\underline{G_{\lambda}^{j_n}}$ *refer to the normalised application 1 under Demanded Resources parameter* λ*. Based on the result, it is shown that 0.50 of x axis hit 0.35 and 0.65 on y axis in Adequate and Sufficient area on y-axis of its respective membership set correspondingly, therefore it is concluded that the fuzzy set for* $\tau_{\omega}\left(\underline{G_{\lambda}^{j_n}}\right) \rightarrow Ra\colon\{0.00,0.00,0.00,1.00,0.00\}$.

In parameter of Processing Time, $\underline{G_{\pi}^{j_n}}$ *is = -0.25.* $\underline{G_{\pi}^{j_n}}$ *refer to the normalised application 1 under Latency Acceptability parameter* π*. Based on the result, it is shown that -0.25 of x axis hit 0.35 and 0.65 on y axis in Minimal and Fair membership sets respectively, therefore it is concluded that the fuzzy set for* $\tau_{\omega}\left(\underline{G_{\pi}^{j_n}}\right) \rightarrow Pt\colon\{0.50,0.50,0.00,0.00,0.00\}$.

Table 4.9: Fuzzy sets after state parameters being normalised

Parameter	**CCG** $(\underline{G_{\varepsilon}^{j_n}})$	**Intercepted Area of Y-axis (Normalised Value)**					$\tau_{\omega}\left(\underline{G_{\varepsilon}^{j_n}}\right)$
Ct	0.20	Extremely Short (0.00)	Short (0.00)	Average (0.55)	Long (0.45)	Extremely Long (0.00)	{0.0,0.0,0.55,0.45,0.00}
Ra	0.50	Critical (0.00)	Limited(0.00)	Adequate (0.00)	Sufficient (1.00)	Plentiful (0.00)	{0.00,0.00,0.00,1.00,0.00}
Pt	-0.25	Minimal (0.00)	Fair (0.50)	Acceptable (0.50)	Efficient (0.00)	Optimal (0.00)	{0.00,0.50,0.50,0.00,0.00}

Fuzzy Inference Module for CCG Calculation

Corresponding fuzzy outputs are determined by mutually comparing fuzzy inputs with the help of fuzzy rules during fuzzy inference similar to the calculation of DoA. However, the fuzzy rules for calculating CCG give higher weight to those approximately impediment state parameters such as long Circulation Time. As a result, the CCG value of the instances indicate more on the limitation instead of the convenience such as adequate Resources Availability and fair Processing Time. The fuzzy rules used to calculate CCG are shown in Figure 4.14 which is based on table 4.6, while the results of fuzzy inputs comparison based on the fuzzy rules are shown in Appendices.

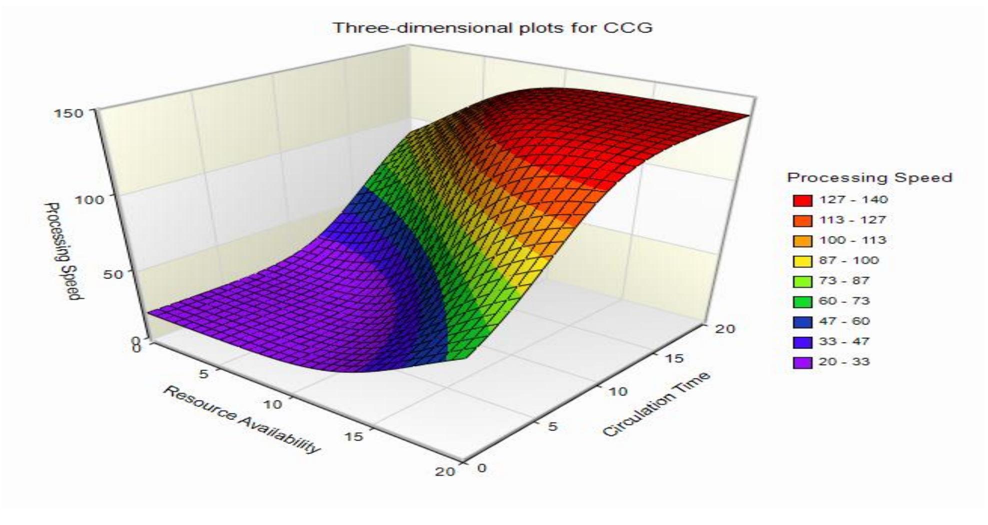
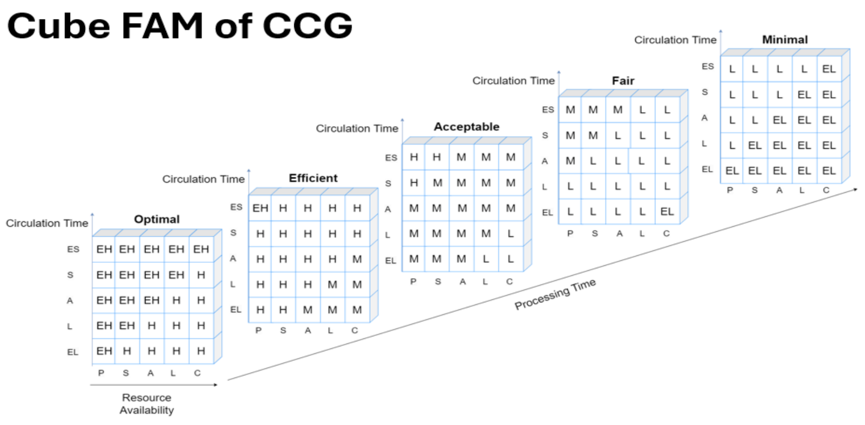


Fig. 4.14: Fuzzy rules for CCG calculation



Optimal	P	S	A	L	C
ES	EH	EH	EH	EH	EH
S	EH	EH	EH	EH	H
A	EH	EH	EH	H	H
L	EH	EH	H	H	H
EL	EH	H	H	H	H

Efficient	P	S	A	L	C
ES	EH	H	H	H	H
S	H	H	H	H	H
A	H	H	H	H	M
L	H	H	H	M	M
EL	H	H	M	M	M

Acceptable	P	S	A	L	C
ES	H	H	M	M	M
S	H	M	M	M	M
A	M	M	M	M	M
L	M	M	M	M	L
EL	M	M	M	L	L

Fair	P	S	A	L	C
ES	M	M	M	L	L
S	M	M	L	L	L
A	M	L	L	L	L
L	L	L	L	L	L
EL	L	L	L	L	EL

Minimal	P	S	A	L	C
ES	L	L	L	L	EL
S	L	L	L	EL	EL
A	L	L	EL	EL	EL
L	L	EL	EL	EL	EL
EL	EL	EL	EL	EL	EL

Fig. 4.15: Fuzzy rules for CCG calculation

From Figure 4.15, the calculation time will be divided into 5 types which are ‘ES’, ‘S’, ‘A’, ‘L’, ‘EL’ and represent ‘Extremely Short’, ‘Short’, ‘Average’, ‘Long’ and ‘Extremely Long’. The resource availability is divided into five types which are ‘P’, ‘S’, ‘A’, ‘L’, ‘C’ and represent ‘Plentiful’, ‘Sufficient’, ‘Adequate’, ‘Limited’, ‘Critical’. The membership degree for each of the fuzzy outputs refers to the lowest membership degree of the compared parameters from the corresponding fuzzy set. The equation used to determine the membership degree is shown in Eq.4.5:

$$\tau_c'(f_{j_n}') = \left(\tau_\theta\left(\underline{K_\theta^{j_n}}\right), \tau_\lambda\left(\underline{K_\lambda^{j_n}}\right), \tau_\pi\left(\underline{K_\pi^{j_n}}\right)\right) \qquad (4.5)$$

Using the result of the comparison of the first fuzzy set as a reference, with the Extremely Short Circulation Time, Critical Resources Available and Minimal Processing Time, the fuzzy output will be Extremely Low. The illustrative explanation is shown in Figure 4.16.

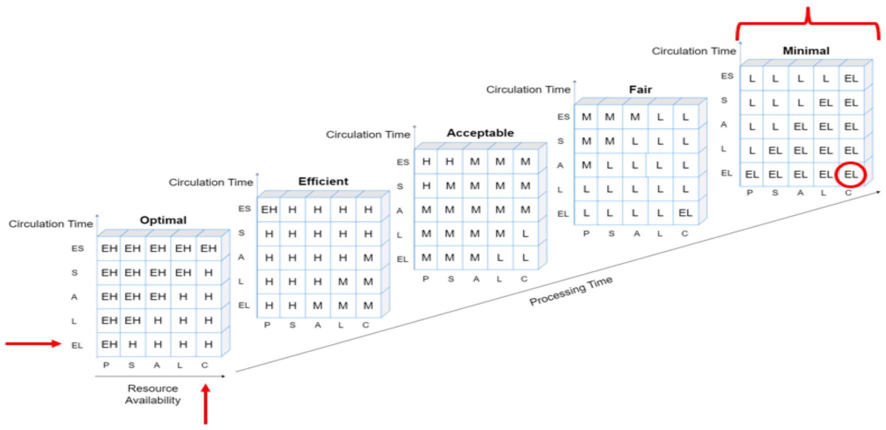


Fig. 4.16: Illustrative explanation on getting fuzzy output for CCG calculation

Defuzzification Module for CCG Calculation

In this case, the singleton value represents the maximum rating of the application for that fuzzy output. The singleton values for fuzzy output are set as ‘Extremely High’, ‘High’, ‘Medium’, ‘Low’ and ‘Extremely Low’ as 10, 8, 6, 4 and 2 respectively. For defuzzification, the membership degrees generated are combined with an equation to obtain the exact CCG, Ω_{j_n} of the instance. The equation used to obtain CCG is shown in Eq.4.6:

$$\Omega_{j_n} = \frac{\sum_{z=1}^{z=u} \quad \tau_c'(f_{j_n}'^{z}) \times \Lambda_z^{f_{j_n}'}}{\sum_{z=1}^{z=u} \quad \tau_c'(f_{j_n}'^{z})} \qquad (4.6)$$

Each membership degree will be multiplied with the corresponding singleton value, then all of the values resulting from the multiplication will be summed up and lastly be divided by the sum of all membership degrees. The CCG obtained is then forwarded to the querying EFN to carry out the following application mapping process.

Application Mapping to Fog Instances

The output of DoA of an application and CCG of a computing instance is named Rating Gain for mapping the application on that instance. The total Rating Gain of the applications are maximised in the process of mapping applications to computing instances. The maximum Rating Gain is able to promote the QoE-aware mapping of the applications. The high combined intensity of associate Assumption Criteria parameters are denoted by the high DoA of the applications. The higher CCG relatively indicates better ability of the instances to satisfy different user assumptions even within the weaknesses. DoA of an application serves as the representative parameter for all of its assumption parameters, therefore the best possible convergence of the assumption parameters to corresponding state parameters of the instances are guaranteed by the maximised Rating Gain of the particular application. As a result, the chance of managing Fog facilities such as computational resources and service accessibility increases without affecting the user assumptions, the QoE regarding the applications are optimised as well.

In an EFN, the applications are mapped to computing instances in the Application mapping unit using a multi-constraint objective function. The multi-constraint objective function is shown in Eq.4.7:

$$max \sum_{\forall e_m \in E_m} \sum_{\forall j_n \in J_n} v_{j_n}^{e_m}\left(\sigma_{e_m} \times \Omega_{j_n}\right) \quad (4.7)$$

Throughout the objective function, the Rating Gain for all application mapping requests are maximised to improve overall user QoE, one to one mapping between applications and instances are guaranteed, and the QoS of the application including service delivery time, service cost and packet loss rate are maintained. In case mapping that satisfies the constraints is not arranged by the EFN, the nodes will be queried for further instances.

The objective function satisfied the decentralised optimization problem. The optimization problem will be solved and the application will be mapped once the application mapping requests are submitted to an EFN. The EFN can solve this optimization problem with multiple constraints using any integer programming solver such as SCIP. A local view of the Fog system is considered by EFN in order to solve the optimization problem. Due to the location, the chance for an EFN to receive a huge load of application mapping requests in a specific time is low. Thus, the optimization problem is less likely to be an NP-hard problem.

Algorithm for QoE-aware Application Mapping

The pseudocode for QoE-aware application mapping is presented in Figure 4.17:

Algorithm 1 QoE-Aware Application Mapping

```
1:    function DoA(bandwidth, demandedResources, latencyAcceptability)
2:    ⟦Bw⟧_ = bandwidth              ← Normalised bandwidth within (-1, 1)
3:    Rr = demandedResources         ← Normalised demanded resources within (-1,1)
4:    Pt = latencyAcceptability      ← Normalised latency acceptability within (-1, 1)
5:    ΓBw = ⟦Bw⟧_                    ← convert Normalised Bw to fuzzy set
```

```
6:      ΓRr = Rr                        ← convert NormalisedRR to fuzzy set
7:      ΓPt= Pt                         ← convert NormalisedPT to fuzzy set
8:      for each ΓBw = min..max do
9:       for each ΓRr = min..max do
10:      for each ΓPt = min..max do
11:       take the largest among the 5 values then
12:       store in Γi
13:       Πs += Γi* s
14:       Π +=  Γi
15:      end
16:      end
17:     end
18:    return DoA = Πs / Π
19      end function
20:     function CCG(circulationTime, availableRequirement, processingTime)
21:     Bw = circulationTime          ← Normalised circulation time within (-1, 1)
22:     Rr = availableRequirement     ← Normalised available requirement within (-1,1)
23:     Pt= processingTime      ← Normalised processing time within (-1, 1)
24:     ΓBw = ⟦Bw⟧_                    ← convert Normalised to fuzzy set
25:     ΓRr = Rr                       ← convert NormalisedRR to fuzzy set
26:    ΓPt= Pt                  ← convert NormalisedPT to fuzzy set
27:     for eachΓBw = min..max do
28:    for each ΓRr= min..max do
29:     for each ΓPt= min..max do
30:      take the smallest among the 5 values then
31:      store in Γi
32:      Πs +=Γi* s
33:     Π+= Γi
34:   end
```

```
35:    end
36:   end
37:     return CCG = Πs /Π
38      end function
```

Fig. 4.17: Pseudocode for QoE-aware application mapping

System Testing for QoE-aware Application Mapping

A policy called Edgeward is implemented in the fog simulator and used as comparison to our proposed solution. Edgeward placement strategy is inclined towards the deployment of application modules close to the edge of the network. However, devices close to the edge of the network may not be computationally powerful enough to host all operators of the application. In such a situation, the strategy iterates on fog devices towards clouds and tries to place remaining operators on alternative devices.

This section is to ensure the validation testing is verifying every class in the iFogSim simulation to be accurate and properly running. Table 4.10 shows some of the simulation parameters which include processing capacity of fog devices, RAM of fog devices, network latency, fog device upstream and downstream capability, module size and tuple size.

Table 4.10: Simulation parameters for QoE-aware application mapping testing

Parameter	**Value**
Processing capacity of fog devices	500 - 4500 MIPS
RAM of fog devices	200 - 1800 Mb
Network latency	2 - 100 ms
Fog Device Upstream capability	1024000 Mbps
Fog Device Downstream capability	1024000 Mbps
Module Size	150 - 1100 MIPS
Tuple Size	100 – 3000 MIPS

Before the comparison between the proposed solutions with others' work is carried out, few scenarios are defined to have an accurate result. In the simulation, few aspects are compared which are the execution time, total power consumption and total network usage. Different aspects will display different results which depend on the scenario. Table 4.11 shows the 6 scenarios used to simulate the results of the proposed solution. All the scenarios are using 1 application to run.

Table 4.11: Simulation scenarios used in the experiments

Scenario	**Fog Device Arrangement**	**Application Module Arrangement**
Scenario 1	Fog Device MIPS increases from end user towards cloud	Module's MIPS requirements increases from client towards last module
Scenario 2	Fog Device MIPS increases from end user towards cloud	Module's MIPS requirements decreases from client towards last module
Scenario 3	Fog Device MIPS increases from end user towards cloud	Module's MIPS requirements is in random order between client and last module
Scenario 4	Fog Device MIPS is in random order between end user and cloud	Module's MIPS requirements increases from client towards last module
Scenario 5	Fog Device MIPS is in random order between end user and cloud	Module's MIPS requirements decreases from client towards last module
Scenario 6	Fog Device MIPS is in random order between end user and cloud	Module's MIPS requirements is in random order between client and last module

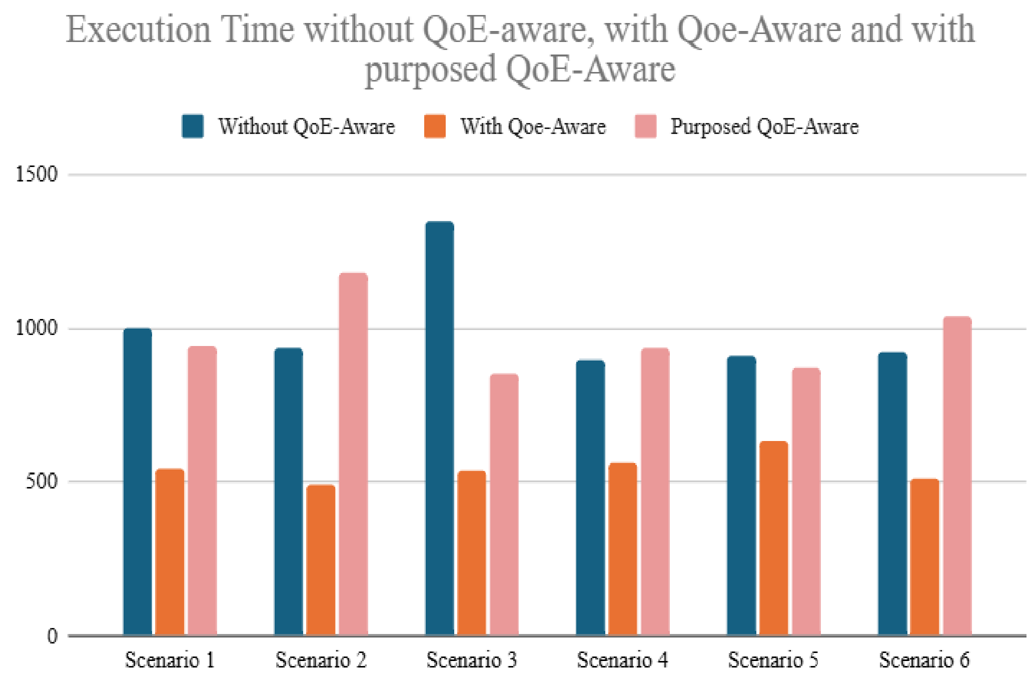


Fig 4.18 and 4.19 shows the comparison of execution time and network usage between application mapping with and without QoE awareness.

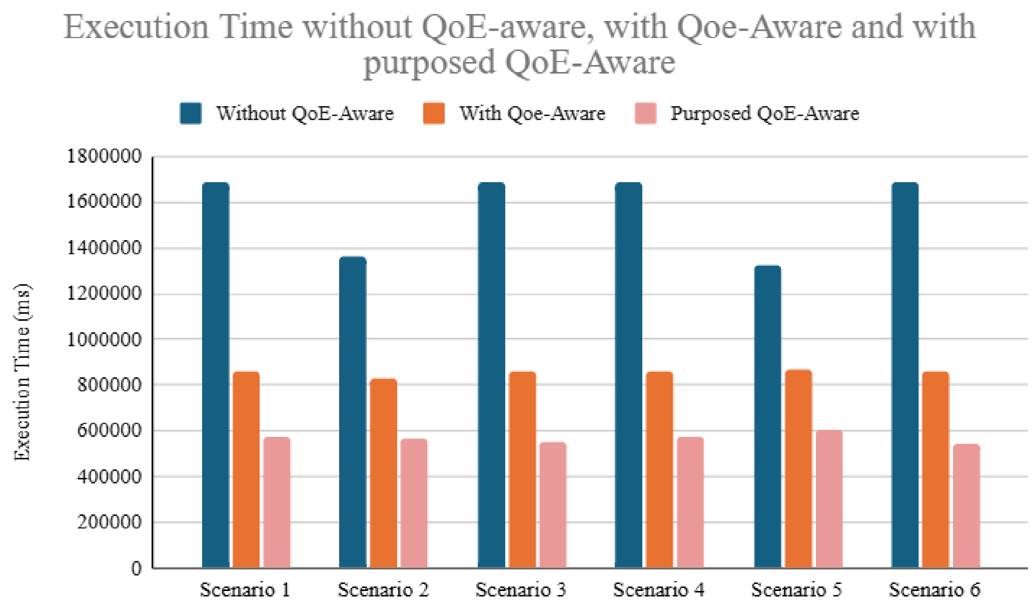
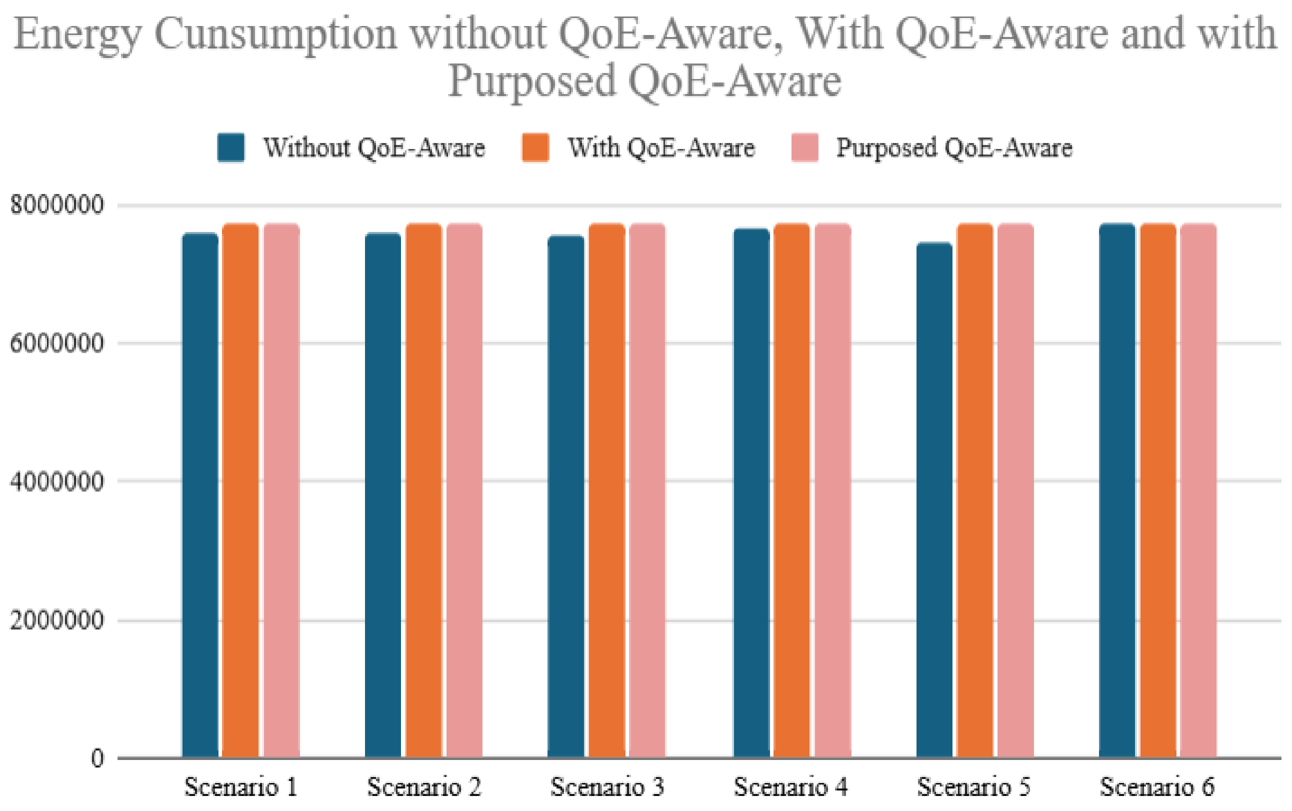


Fig. 4.18: Bar Chart for execution time for solution with and without QoE-aware and with proposed QoE-aware application allocation with 1 and 5 application

Figure The bar chart in Figure 4.18 compares the execution times for three approaches: without QoE-aware, with QoE-aware, and the proposed QoE-aware algorithm in 6 different scenarios. In the figure, the x-axis shows the different scenarios while the y-axis represents the execution time in milliseconds. Besides on figure 4.18 where each solution's execution time are significantly different, it is illustrated that the execution time of the solution with QoE-aware is clearly shorter than without QoE-aware and proposed QoE-Aware. Hence, it showed that the QoE-Aware can fulfil the user requirement on QoE (improve the user satisfaction) and at the same time **reduce** the execution time. However, the proposed QoE-Aware application allocation is having high execution time compared to previous QoE-aware solutions when executing 1 application. When we add the number of application to 5, we can see that the execution time of purposed QoE-aware application allocation are having significant reduction.



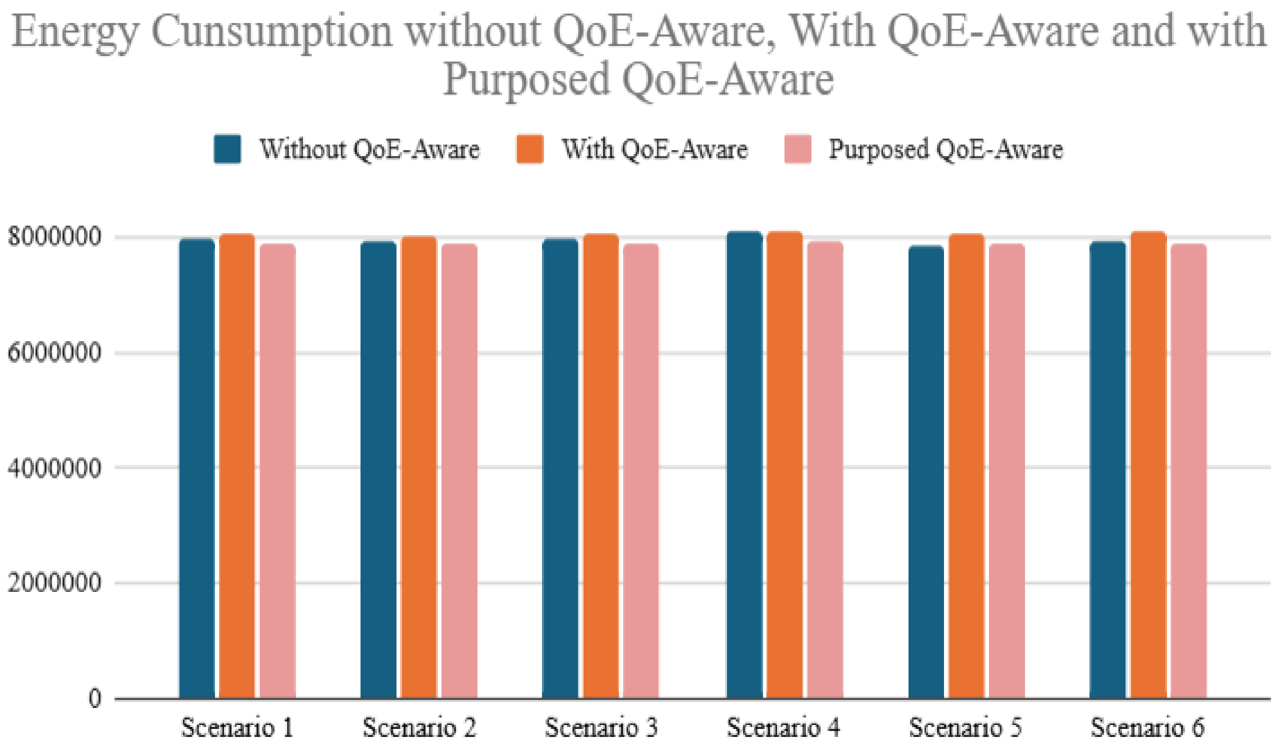


Fig. 4.19: Bar chart for total energy consumption of application with and without QoE-aware

Figure 4.19 shows the total energy consumption of application in 6 different scenarios by using QoE-aware and without QoE-aware. In the figure, the x-axis represents the difference of scenarios while the y-axis shows the total energy consumption. Based on the figure above, the total energy consumption of the application with QoE-aware is **slightly higher** than without QoE-aware. This is because to fulfil the QoE requirement, the energy consumption will increase to carry out the QoE-aware module placement. In short, although energy consumption has not been improved, the QoE requirement is fulfilled.

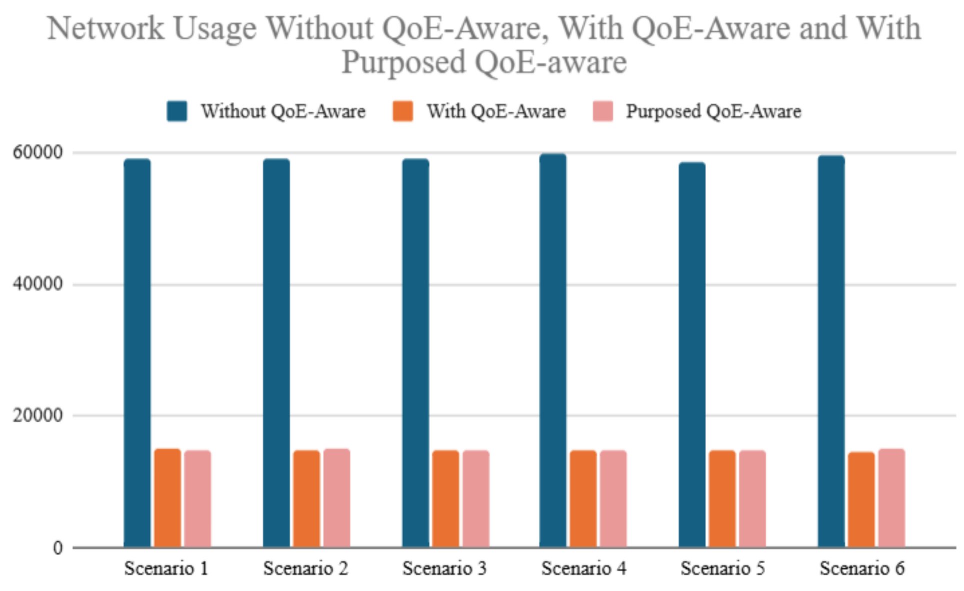


Fig. 4.20: Bar chart for comparison of network usage for solution with and without QoE-aware

Figure 4.20 shows the total network usage of the solution with and without QoE-aware in 6 different scenarios. In the figure, the x-axis represents the different types of scenarios while the y-axis shows the total network usage. In the 6 scenarios, the results show that the solution with QoE-aware is better as it has the **lower** network usage compared to the solution without QoE-aware. This is because the application did not run or use the cloud but just using the fog computing so the network usage is much lower. It indicates that it can fulfil the user requirements on QoE and at the same time reduce the network usage.

Energy-Aware Module Placement

The energy-aware module placement is the second process in phase one of the proposed solution, the objective is to optimise the energy consumption. The reason for proposing the Energy-aware is to reduce the energy consumption in the fog computing.

Analytical Modeling of Proposed Energy-aware Algorithm

To place the modules, a minimum energy requirement by a module was estimated then placed into the fog device that can handle the modules.

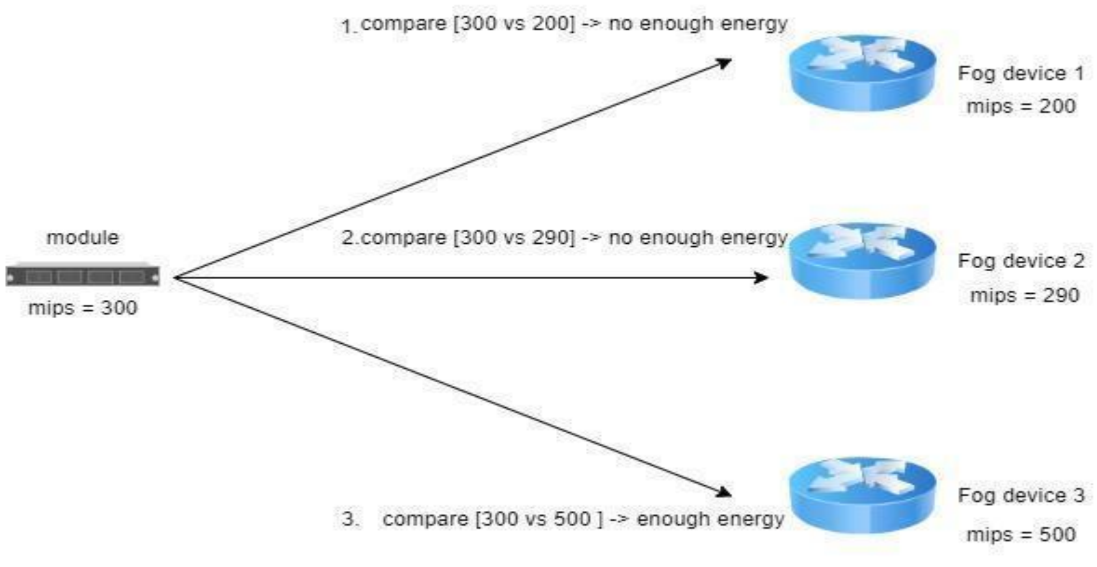

1. compare [300 vs 200] -> no enough energy
Fog device 1
mips = 200
module
mips = 300
2.compare [300 vs 290] -> no enough energy
Fog device 2
mips = 290
3. compare [300 vs 500] -> enough energy
Fog device 3
mips = 500

Fig. 4.21: Processing of energy-aware module placement

Figure 4.21 shows that the process of placing the incoming module and above is the example of the process. As the figure showed the module minimum energy needed and MIPS, there are 1,2,3.....n fog devices in which are represented by different maximum energy and MIPS. At first, if the fog device is not enough maximum energy and MIPS for the incoming module, the fog device will forward the module to the following fog device and find a fog device which can fulfil the requirement of the module.

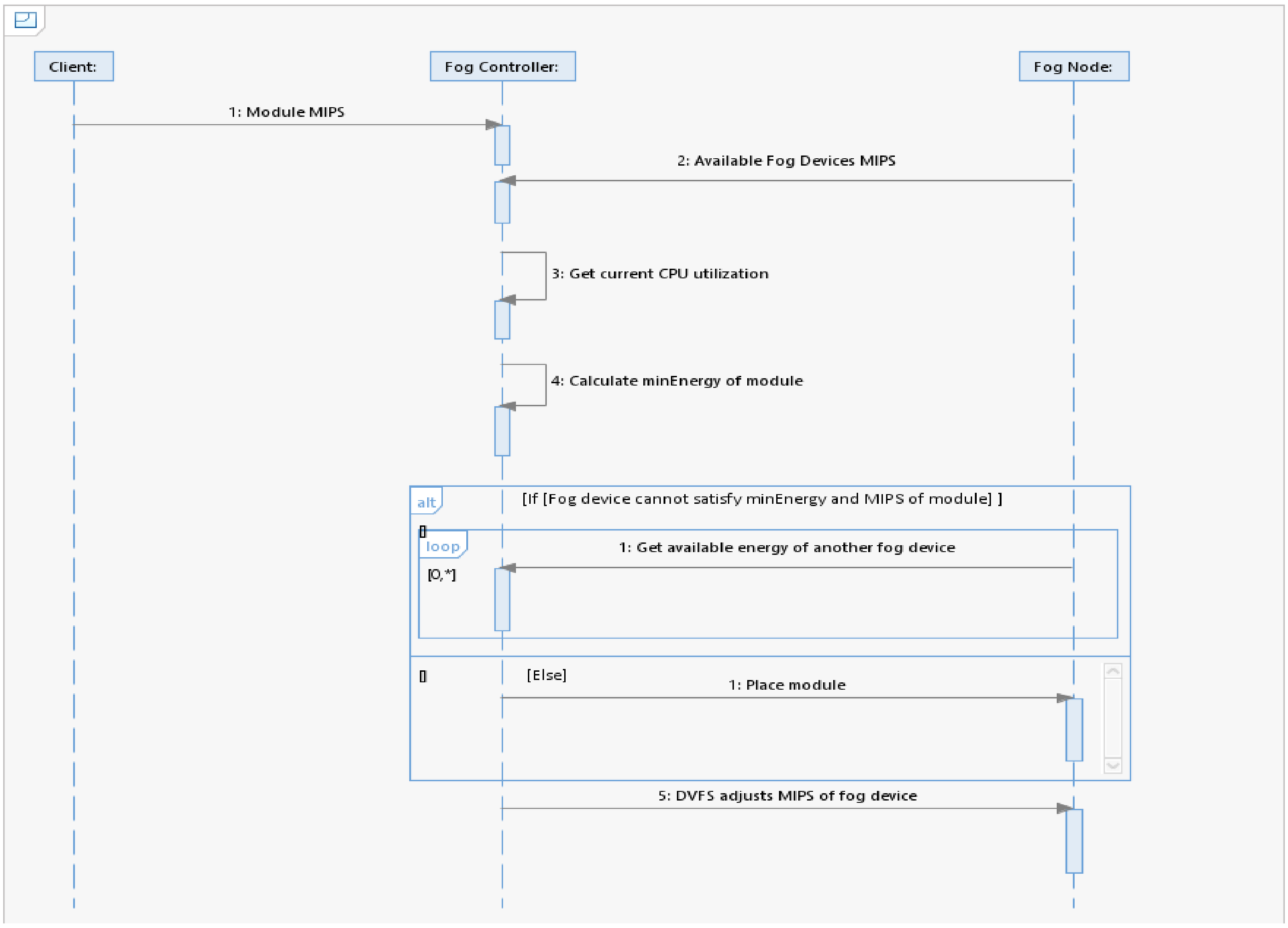

Client:
Fog Controller:
Fog Node:
1: Module MIPS
2: Available Fog Devices MIPS
3: Get current CPU utilization
4: Calculate minEnergy of module
alt
[If [Fog device cannot satisfy minEnergy and MIPS of module]]
loop
[0,*]
1: Get available energy of another fog device
[Else]
1: Place module
5: DVFS adjusts MIPS of fog device

Fig. 4.22: Sequence diagram for energy-aware module placement

Figure 4.22 shows the sequence diagram for energy-aware module placement that illustrates the essential steps in achieving energy saving in fog computing. The first step is to get the MIPS of the incoming module and available fog devices which is the execution speed of the computer's CPU. Next is to get the current CPU utilisation based on the two MIPS values. After that, minimum energy needed by that incoming module is calculated. Once it is calculated, the MIPS and minimum energy of the module is compared with the available energy of the fog device until a suitable fog device is found and eventually the module will be placed to that fog device. Lastly, the DVFS will adjust the MIPS of the fog device into a value of used MIPS value.

Parameter	**Definition**
Pmax	Maximum power
Ps	Static power
Pc	Constants power
U	Utilisation
M mips	The MIPS of the incoming module
F mips	The available fog devices MIPS
Min Energy	The get the minimum energy of incoming module
frequency	The available frequency value

maxFre	The max frequency number on HOST (in GHz)
N_{CV}	Closest value of energy consumption fog devices
A_M	Module
E_{CE}	Estimated consumed energy after allocation
A_{MIPS}	Adjusted MIPS = $\frac{Fmips}{MaxFre}$ ×frequency [χ]
N_{NAME}	Name of Fog Device
VR_{NAME}	VRGame's name
C	Client
P_{DVFS}	Process of Dynamic voltage and frequency scaling
U	Utilisation

The following formula is applied to perform the energy-aware module placement.

$$\frac{P_{max} - P_s}{100} \tag{4.8}$$

Using P_{max}- P_s and divided by 100, in order to obtain the constant power value.

$$minEnergy = (P_s + P_c) \times U \times 100 \tag{4.9}$$

In this formula, the formula above is used to do the calculation and get the estimated minimum energy of the module.

$$U = Min\left(1, \frac{M_{mips}}{F_{mips}}\right) \tag{4.10}$$

The parameter U shown above is the parameter put inside the Eq.4.9. The formula for the parameter U is using the function Math.min and compare the 1 and the result which is $\frac{M_{mips}}{F_{mips}}$. In other word, the Eq.4.10 formula was used to obtain the current utilisation of CPU from and applied by the *minEnergy* and compare with the available energy of fog devices. The process will loop until the module is found the fog device which satisfy the $minEnergy$ and M_{mips} of the module.

After the modules are placed to the fog devices, DVFS is performed in order to check whether the fog devices still have available resources or not. If the fog device still has plenty of remaining MIPS, then DVFS will adjust the MIPS of the fog device into a value that is close to the used MIPS value.

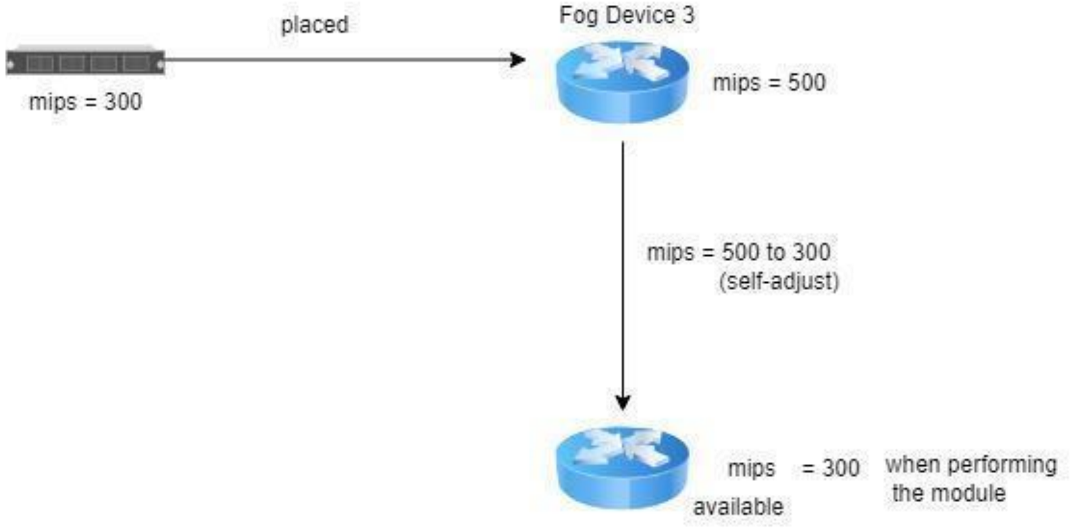


Fig. 4.23: Processing of Dynamic voltage and frequency scaling (DVFS)

DVFS will adjust the MIPS of the selected fog device. After that, the selected fog device MIPS is adjusted as close as possible with the incoming module MIPS.

Algorithm 2 Energy-Aware Module Placement

```
1: function $N_{CV}$ ($F_{MIPS}$, $P_{MAX}$, $N_{NAME}$)
2: for each $N_{CV}$ do
3:     for each $A_M$ do
4:     function U
5:     if U($N_{CV}$) bigger than $M_{MIPS}$(minEnergy) then
6:     add $M_{MIPS}$ to $N_{CV}$
7:     return $N_{NAME}$
8:     deployed modules
9:     end if
10:    if ($N_{NAME}$ equal $VR_{NAME}$ AND $N_{NAME}$ equal C) then
11:    add $M_{MIPS}$ to $N_{CV}$
12:    return $N_{NAME}$
13:    deployed modules
14:    end if
15:    end
16:    end
17:    for each $N_{CV}$ do
18:    if $N_{NAME}$ include "cloudlet" then
19:    $P_{DVFS}$($N_{CV}$)
20:    end if
21: end
```

Fig. 4.24: Pseudocode for energy-aware module placement

For any fog device that is closest, it will perform the module of estimating the consumed energy after allocation. If the device is suitable for the module placement, then it will add the module to the device and print the device name before the modules are deployed, else it will find the upper level of devices for suitable module placement. If the device name equals m-VRGame and the module name is client, then it will add the module to the device and print the device name before the modules are deployed.

Algorithm 3 DVFS (Dynamic Voltage and Frequency Scaling

```
1:      used mips = fog device total MIPS – fog devices available MIPS
2:      Device adjust total mips = used mips
```

Fig. 4.25: Pseudocode for DVFS

First, calculate the used mips by using the formula. After that, the fog device total MIPS is set to equal to the used MIPS. For the scenarios used to test the result of the proposed algorithms, please refer to Table 4.13.

Table 4.13: Simulation parameters for energy efficiency application mapping testing

Parameter	**Value**
Processing capacity of fog devices	2500 - 6500 MIPS
RAM of fog devices	200 - 1800 Mb
Network latency	2 - 100 ms
Fog Device Upstream capability	1024000 Mbps
Fog Device Downstream capability	1024000 Mbps
Module Size	150 - 1100 MIPS
Tuple Size	100 – 3000 MIPS

System Testing For Energy Efficiency

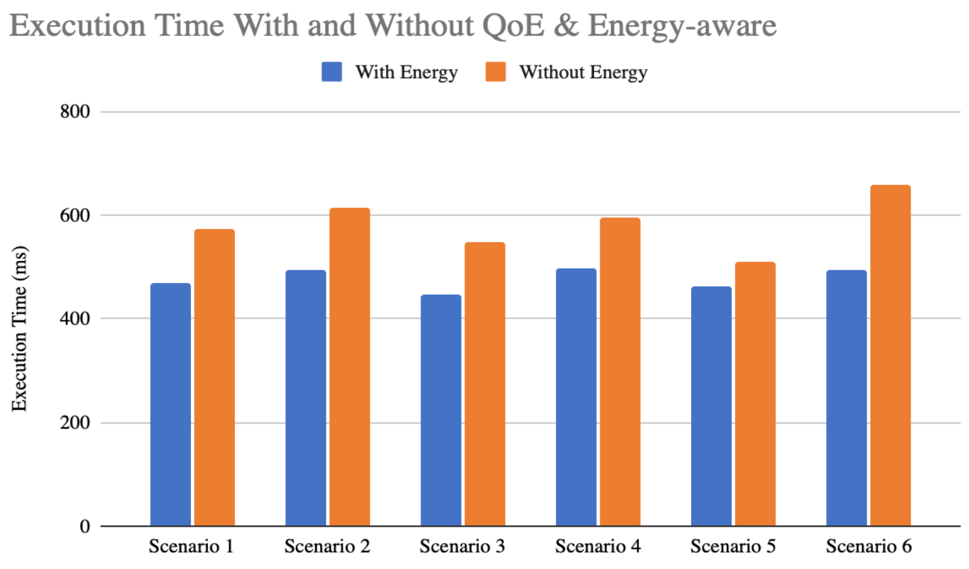


Fig. 4.26: Execution time of application with and without QoE & Energy-aware

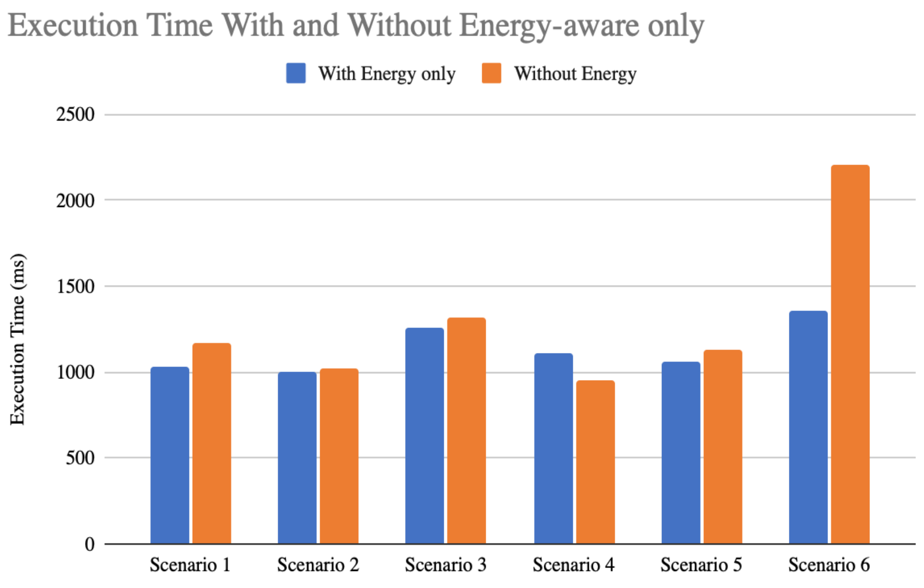


Fig. 4.27: Execution time of application with and without Energy-aware only

Figure 4.26 shows the execution time of the application with and without QoE & Energy-aware in different scenarios. In the figure, the x-axis represents the difference of scenarios while y-axis shows the execution time in milliseconds. The result shows that the execution time of the application with QoE and Energy-aware is shorter than without QoE and Energy-aware. As a result, the QoE and Energy-aware placement is able to fulfil the QoE requirements and further reduce the execution time as compared to application without QoE and Energy-aware.

Figure 4.27 shows the execution time of the application with and without Energy aware in different scenarios. In the figure, the x-axis represents the 6 differences of scenarios; the y-axis shows the execution time in milliseconds. As the graph above it can be clearly seen that the results of the execution time with Energy-aware only is consistent throughout the 6 scenarios and has a lower execution time as compared to without Energy-aware except for scenario 4 whereas the execution time without Energy-aware is inconsistent. Overally, it can be concluded that by implementing the Energy-aware only can reduce the execution time too.

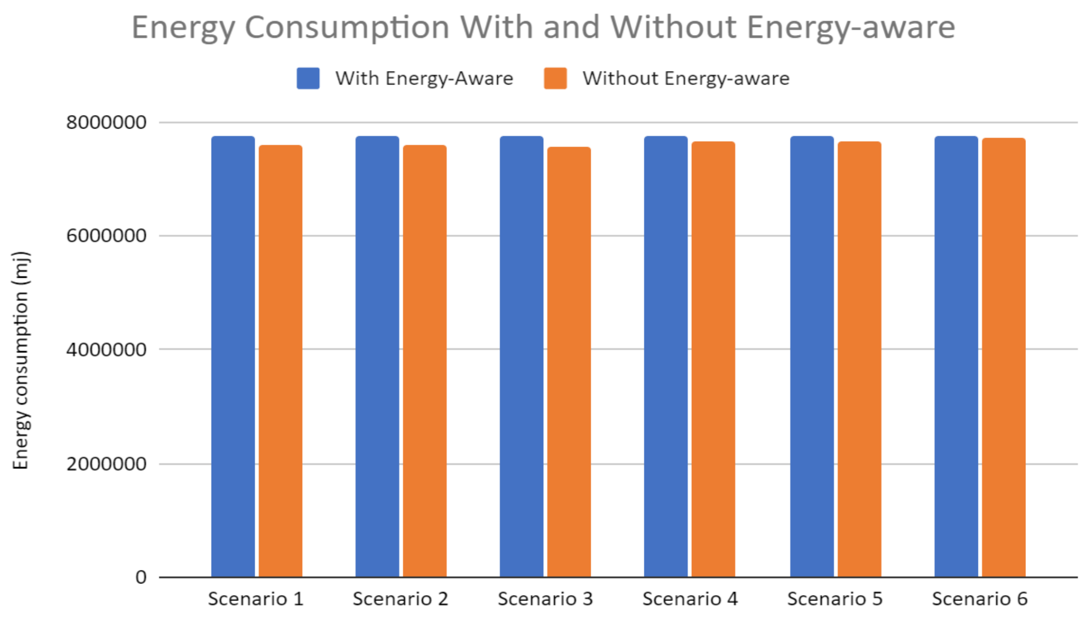


Fig. 4.28: Total energy consumption of application with and without QoE & Energy-aware

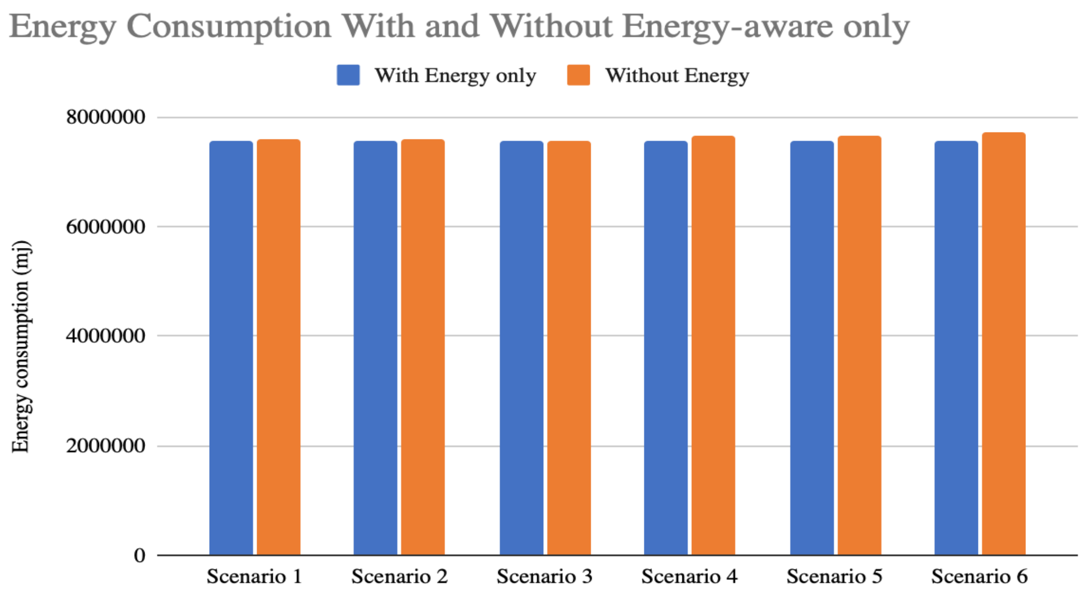


Fig. 4.29: Total energy consumption of application with and without Energy-aware only

Figure 4.28 shows the total energy consumption of application in 6 different scenarios by comparing QoE, Energy-aware with without Energy-aware. In the figure, the x-axis represents the 6 different scenarios while the y-axis shows the total energy consumption in a megajoule. Based on the figure above, the total energy consumption of the application with QoE and Energy-aware is slightly higher than the application without QoE and Energy-aware. This result stated that energy consumption is not reduced when at the same time fulfilling the QoE requirement.

Figure 4.29 shows the total energy consumption of application in 6 different scenarios by comparing Energy-aware only with without Energy-aware. In the figure, the x-axis represents the 6 different scenarios while the y-axis shows the total energy consumption in a megajoule. According to the figure, it shows that the total energy consumption is reduced.

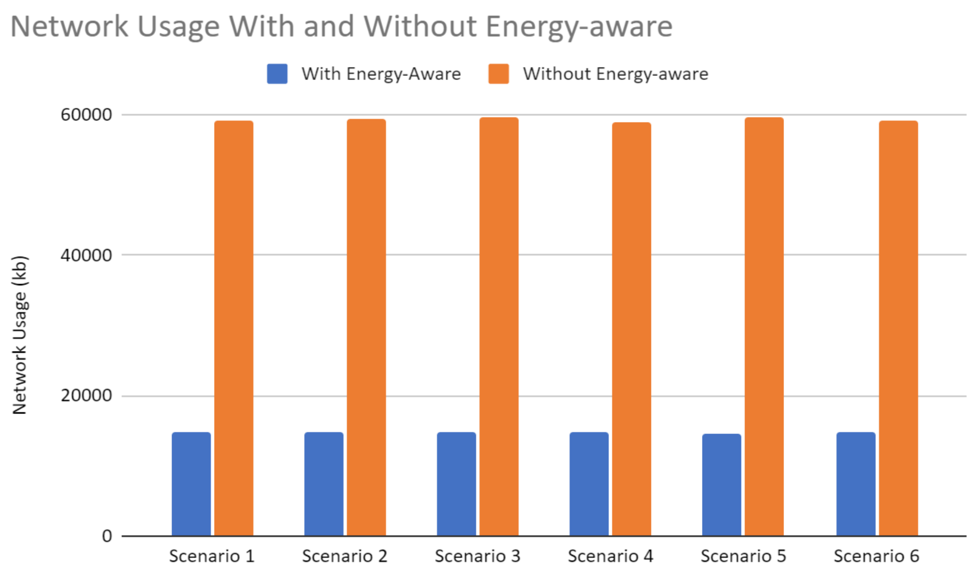


Fig. 4.30: Total network usage of application with and without QoE & Energy-aware

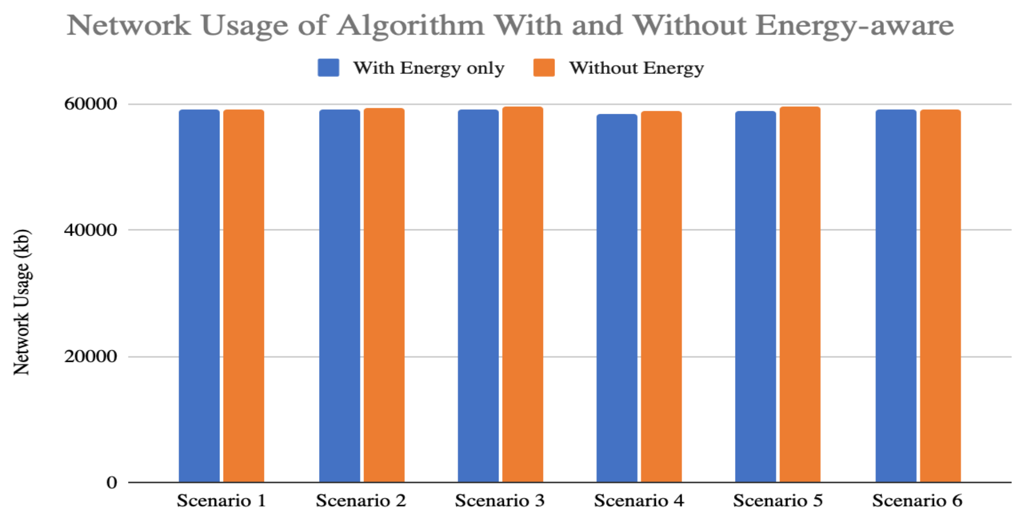


Fig. 4.31: Total network usage of application with and without Energy-aware only

Figure 4.30 shows the results of total network usage of application with and without QoE, Energy-aware in 6 different scenarios. In scenario 3 and 5, the total network usage with QoE and Energy-aware is lower compared to the network usage without QoE and Energy-aware. This result concluded that the total network usage of applications with QoE and Energy-aware will be better compared to without QoE and Energy-aware.

Figure 4.31 shows the results of total network usage of applications with and without Energy-aware only in 6 different scenarios. In all scenarios, the total network usage with Energy-aware only is lower compared to the network usage without Energy-aware. This result concluded that the total network usage of applications with Energy-aware only will be better compared to without Energy-aware.

The result showed the parameter of the infrastructure and application that the team defined to run the stimulation of the energy algorithm in iFogsim. According to the figure, the result of the Energy-aware algorithm is shown. Through the result, the team examines the total network usage, cost of execution in cloud, total energy consumption, energy consumed, etc. As a result, the team can use these results to compare the proposed algorithm with other energy aware algorithms to determine whether the proposed algorithm has the better capability and performance.

Proposed Offloading Algorithm

The result showed the parameter of the infrastructure and application that the team defined to run the stimulation of the energy algorithm in iFogsim. According to the figure, the result of the Energy-aware algorithm is shown. Through the result, the team examines the total network usage, cost of execution in cloud, total energy consumption, energy consumed, etc. As a result, the team can use these results to compare the proposed algorithm with other energy aware algorithms to determine whether the proposed algorithm has the better capability and performance.

In phase two of the proposed solution, the proposed Offloading Algorithm is implemented. It is assumed that the fog layer consists of large amounts of fog devices that can host the application for more than one instance. First, the MIPS of the fog devices are determined. When a new task arrives at the fog device, the MIPS of the new task from the application will be compared with the current MIPS of the fog device. If the current fog device is already processing a task, this means the fog device will have higher MIPS compared to other fog devices. Therefore, the system will offload the task to another fog device to prevent the fog device from being fully scheduled and affecting the performance.

The distance between the fog device that is executing the job and the end user's device determines the network usage. For instance, the further the fog device is located from the user, the higher the network usage due to the fact that the job has to travel through a lot of fog devices in order to reach the destination fog device that is responsible for executing the job.

The total number of modules to execute determines the execution time. For instance, the higher the number of modules to be executed on a fog device, the higher the execution time due to the fact that the job has to wait for resources from the resource constrained fog device that are currently overloaded with modules.

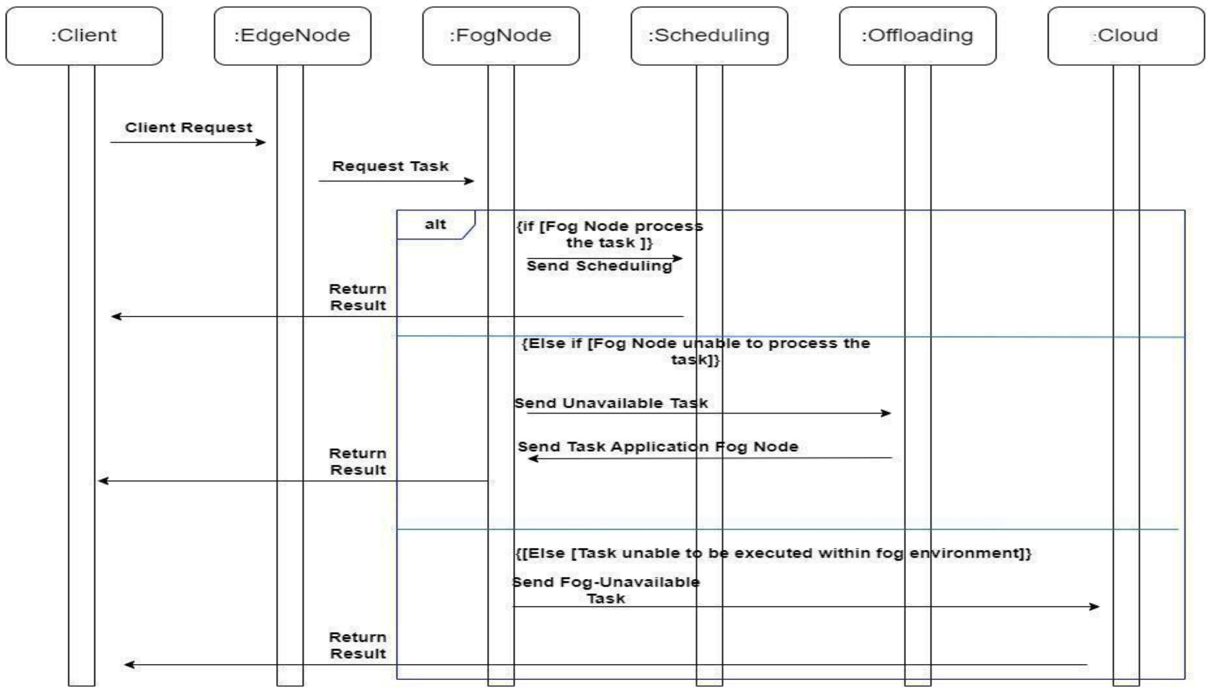

:Client
:EdgeNode
:FogNode
:Scheduling
:Offloading
:Cloud
Client Request
Request Task
alt
{if [Fog Node process the task]}
Send Scheduling
Return Result
{Else if [Fog Node unable to process the task]}
Send Unavailable Task
Send Task Application Fog Node
Return Result
{[Else [Task unable to be executed within fog environment]}
Send Fog-Unavailable Task
Return Result

Fig. 4.32: Sequence diagram for offloading

Figure 4.32 shows the sequence diagram for offloading. Firstly, the client will send a task execution request to the edge node and the task execution request will be forwarded to the fog node. The maximum job load for each fog node is determined. If the fog node is able to execute the task, the task will be scheduled for execution, otherwise the task will be offloaded to another applicable fog node. There is also a situation that the task cannot be executed within the fog environment and the task will be forwarded to cloud in this case. After the task has been executed, all the results will be sent back to the client. Figure 4.33 shows the pseudocode of the Computation Offloading algorithm.

Algorithm 4 Computation Offloading algorithm

```
1: function ResourceDiscovery(NF, FG, DEND, A)
2: for each FD in NF do
3: FOGmips = FOGmips + Dmips
4: end for
5: for each M in NM do
6: APPmips = APPmips + Mmips
7: end for
8: Initialize MMAP
9: NMP = QoEApplicationMapping(app)
10: NFC = EnergyModulePlacement(NF, FG, DEND)
11: while FOGmips > APPmips do
12: FOGmips= MapDeviceLoop(NFC, NMP, MMAP)
13: end while
14: end function
```

Fig. 4.33: Pseudocode of proposed Offloading algorithm

System Testing for Proposed Offloading Algorithm

The simulation parameters are similar with those in the system testing conducted for QoE-aware mapping which includes processing capacity of fog devices, RAM of fog devices, network latency, fog device upstream capability, fog device downstream capability, module size and tuple size. The simulation parameters are shown in Table 4.14. Strategies with and without the proposed Offloading Algorithm are tested according to the parameter value are shown in Table 4.14.

Table 4.14: Simulation Parameters for Computation Offloading algorithm testing

Parameter	**Value**
Processing capacity of fog devices	2500 - 6500 MIPS
RAM of fog devices	200 - 1800 Mb
Network latency	2 - 100 ms
Fog Device Upstream capability	1024000 Mbps
Fog Device Downstream capability	1024000 Mbps
Module Size	150 - 1100 MIPS
Tuple Size	100 – 3000 MIPS

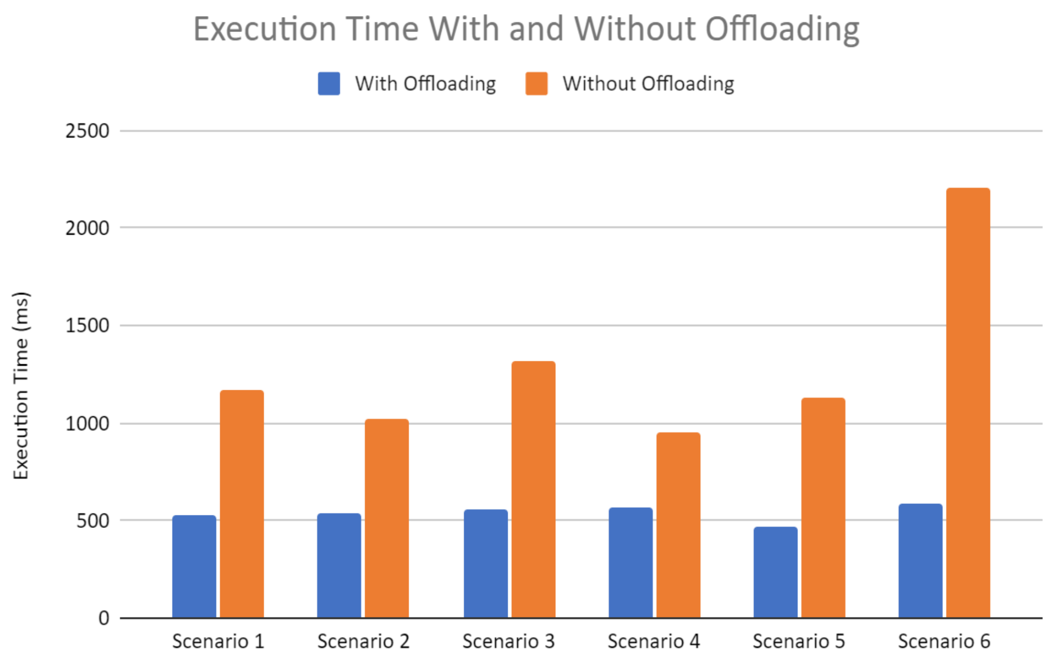


Fig. 4.34: Bar chart for comparison of execution time algorithm with and without QoE-aware & Energy-aware & Offloading

Figure 4.34 shows the execution time of the solution with and without the proposed Offloading algorithm in 6 different scenarios. In the figure, the x-axis represents the difference of scenarios while the y-axis shows the execution time in milliseconds. By looking at the chart, it is shown that there is much difference in the execution time in different scenarios for both solutions, while it is clearly visible that the solution with offloading has decreased the execution time.

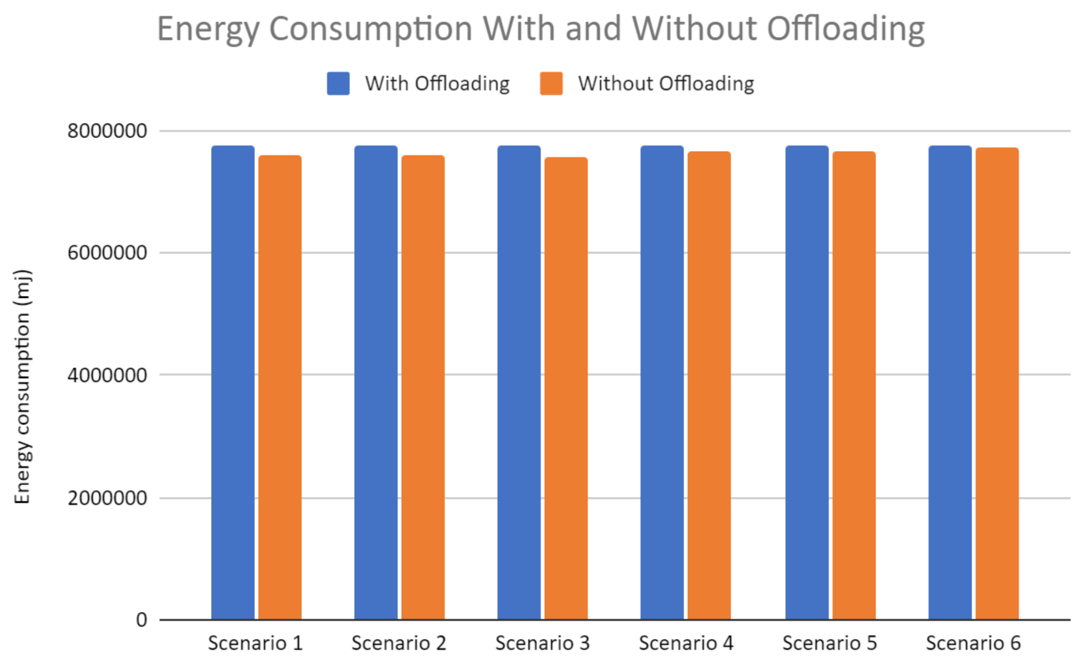


Fig. 4.35: Energy consumption of algorithm with and without QoE & Energy-aware & Offloading

Figure 4.35 shows the total energy consumption of application in 6 different scenarios by using the proposed offloading algorithm and without the proposed offloading algorithm. In the figure, the x-axis represents the difference of scenarios while the y-axis shows the total energy consumption in megajoules. Based on the figure above, the total energy consumption of the application with QoE, Energy-aware and proposed offloading algorithm is equal to without computation offloading algorithm. This result already stated clearly that the proposed offloading algorithm would increase the energy consumption.

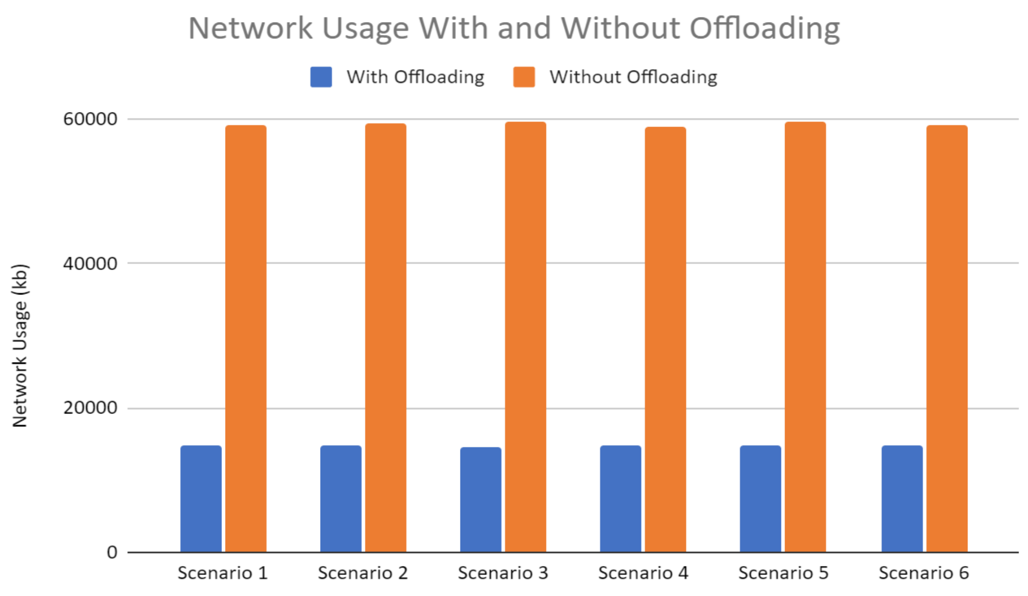


Fig. 4.36: Bar chart for comparison of network usage for solution with and without enhanced Offloading algorithm

Figure 4.36 shows the total network of the solution with and without QoE, Energy-aware and proposed offloading algorithm in 6 different scenarios. In the figure, the x-axis represents the difference of scenarios while the y-axis shows the total network usage in milliseconds. By looking at the chart, it is concluded that the solution with the QoE, Energy-aware and proposed Offloading algorithm has **lower** network usage

Chapter Summary and Evaluation

In this chapter, the QoE-aware application mapping implemented has proven able to reduce the execution time and network usage but will increase the energy consumption. The implementation of QoE-aware application mapping with energy-aware module placement is able to further reduce the execution time and network usage but energy consumption was not reduced. However, implementing energy-aware module placement only, the execution time, energy consumption and network usage will be slightly reduced. Hence, QoE-aware and energy-aware cannot be guaranteed at the same time. Furthermore, implementing QoE-aware application mapping, energy-aware module placement with offloading algorithms will slightly reduce the network usage but will increase in execution time and have no effect on the energy consumption. In short, the proposed algorithms can fulfil the user QoE requirement and at the same time would not overload the fog devices. Energy consumption can be reduced if solely implementing the energy-aware module placement.

Evaluation

In this chapter, the QoE-aware application mapping implemented has proven able to reduce the execution time and network usage but will increase the energy consumption. The implementation of QoE-aware application mapping with energy-aware module placement is able to further reduce the execution time and network usage but energy consumption was not reduced. However, implementing energy-aware module placement only, the execution time, energy consumption and network usage will be slightly reduced. Hence, QoE-aware and energy-aware cannot be guaranteed at the same time. Furthermore, implementing QoE-aware application mapping, energy-aware module placement with offloading algorithms will slightly reduce the network usage but will increase in execution time and have no effect on the energy consumption. In short, the proposed algorithms can fulfil the user QoE requirement and at the same time would not overload the fog devices. Energy consumption can be reduced if solely implementing the energy-aware module placement.

This chapter will be focused on the proposed solution evaluation as well as the data collection technique used to perform analysis on the result of the QoE, Energy-Aware and Offloading algorithm. Next, A statistical model is used to ensure the accuracy and consistency of the data. Other than that, experiment setup and several tools are used in order to analyse and evaluate the QoE, Energy-Aware and Offloading algorithm performance.

First and foremost, the evaluation of the network usage, consumption of power and execution time based on the results obtained from the proposed QoE application allocation, QoE algorithm and Edgeward begins. Next, the Energy-Aware algorithm will then be implemented into the proposed QoE application allocation and QoE algorithm and the result will be tested and compared with the Edgeward. Thirdly, the enhanced algorithm will be implemented into the QoE-Aware and Energy-Aware algorithm and compared with the Edgeward.

This chapter is organised as follows: Section 5.1 describes performance setup that includes the experiment setup and components. Section 5.2 describes the data collection method that we used to collect the data. Section 5.3 presents the statistical model that we used to ensure the accuracy of results collected. Section 5.4 discusses the performance analysis of proposed QoE-Aware algorithm, QoE-Aware algorithm added with Energy-Aware algorithm and together with the proposed Offloading algorithm. Finally, Section 5.5 will be the conclusion of the chapter.

Performance Setup

In order to evaluate the performance of the proposed algorithm, a predefined set of fog computing resource and application module job parameters are used and shown in Table 5.1. A fog-cloud environment is modelled in a way that it consists of a total number of nine fog devices in the fog layer. The unit used to measure the processing power is the million instructions per second (MIPS). Next, as the MIPS of a fog device is high, then the better or faster the fog device in handling and efficiently performing the tasks.

Table 5.1: A predefined set of fog computing resources and application module parameters

Parameter	Value
Processing capacity of fog devices	350 - 1000 MIPS
Ram of fog devices	256 - 512 Mb
Network latency	2 - 100 ms
Fog Device Upstream capability	10000 - 1024000 Mbps
Fog Device Downstream capability	10000 - 1024000 Mbps
Module Size	100 - 600 MIPS
Tuple Size	1000 - 6000 MIPS

Several application modules are divided to enable those modules to be hosted by individual fog devices in the fog layer for the purpose to process the job of the application modules. Besides that, five applications are created and divided into four modules for each application to be hosted in the fog layer for the testing simulation purposes. Next, 100 MIPS to 600 MIPS are the variation requirement for each application module.

The pseudocode for Edgeward module placement is presented in Figure 5.1:

Algorithm 1: Edgeward module placement

```
1: for p ∈ PATHS do Across all paths
2:     placedModules ←{};
3:     for Fog device d ∈ p do                    ⇾ leaf-to-root traversal
4:     modulesToPlace ←{};
5:         for module w ∈ app do                  ⇾ find modules ready for placement on device d
```

```
6:              if all predecessors of w are in placedModules then        ⇾ if all predecessors are placed
7:                  add w to modulesToPlace;
8:              end
9:          end
10:         for module θ ∈ modulesToPlace do
11:             if d already has instance of θ as θ' then
12:                 if $CPU_{\theta}^{req} \geq CPU_{d}^{avail}$ then        ⇾ device d does not have CPU capacity to host θ
13:                     $\underline{\theta}$ ←merge(θ, θ');
14:                     $f$ ←parent(d);
15:                     while $CPU_{\theta}^{req} \geq CPU_{f}^{avail}$ do        ⇾ find device north of d for hosting θ
16:                         $f$ ←parent($f$);
17:                     end
18:                 Place $\underline{\theta}$ on device $f$;    ⇾ device d can host θ
19:                     add θ to placedModules;
20:                 end
21:                 else
22:                 Place θ on device d;
23:                     add θ to placedModules;
24:                 end
25:             end
26:             else if no device north of d has an instance of θ then
27:             if $CPU_{\theta}^{req} \leq CPU_{d}^{avail}$ then        ⇾ if not, will be handled by subsequent iterations
28:                 Place θ on device d;
29:                     add θ to placedModules;
30:                 end
31:                 else
32:                     $f$ ←parent(d);
```

```
33:                         while $CPU_{\theta}^{req} \geq CPU_{f}^{avail}$ do      ⇾ find device north of d for hosting $\theta$
34:                             $f$ ←parent($f$);
35:                         end
36:                     Place $\theta$ on device $f$;     ⇾ device f can host $\theta$
37:                         add θ to placedModules;
38:                     end
39:                 end
40:             end
41:         end
42:     end
```

Fig 5.1: Pseudocode for Edgeward Module Placement

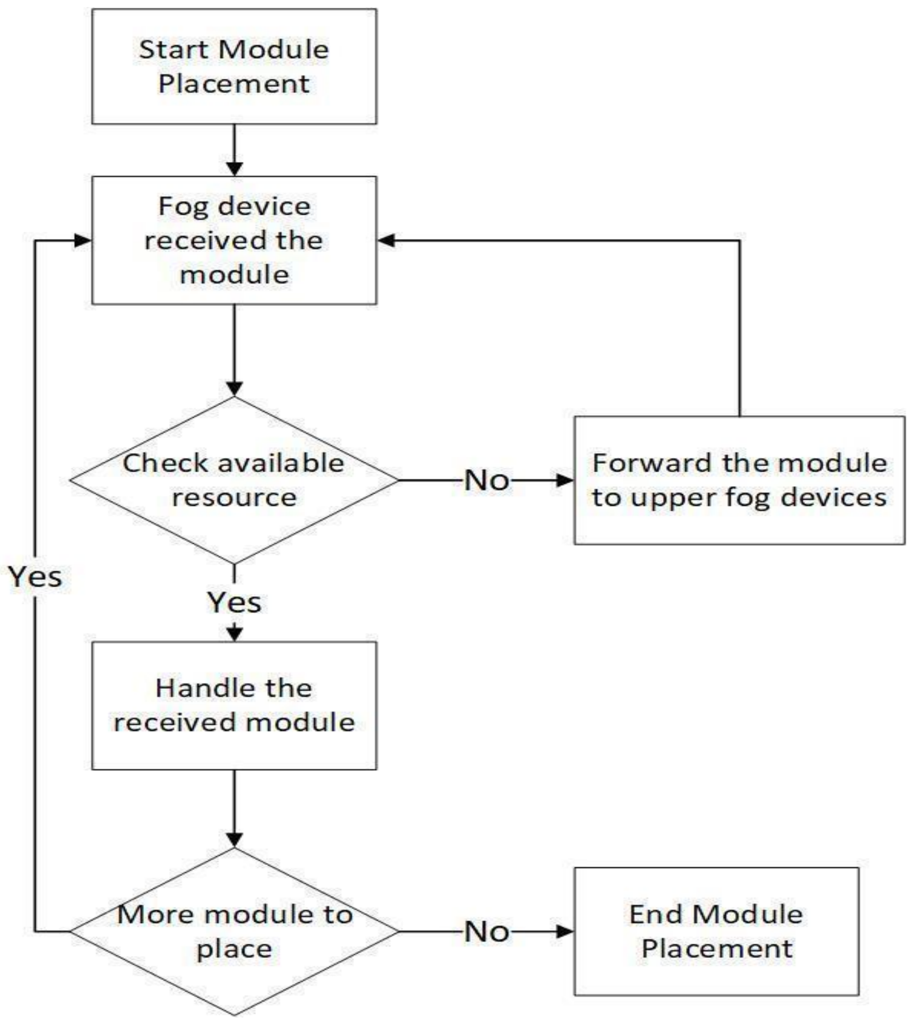


Fig 5.2: Edgeward Module Placement Process

According to Figure 5.1 and 5.2, all the modules will be sent to the fog devices which are closest to the edge network, also known as users' network. When the module reaches the fog device, the fog device will check on its own available resources in order to determine if the module could be placed on itself. The modules will be always placed on the nearest fog devices until the fog devices are out of resources. If so, the rest of the modules will be forwarded to the upper fog devices which are closer to the cloud. The process of the Edgeward Module Placement will be iterated until there is no more module.

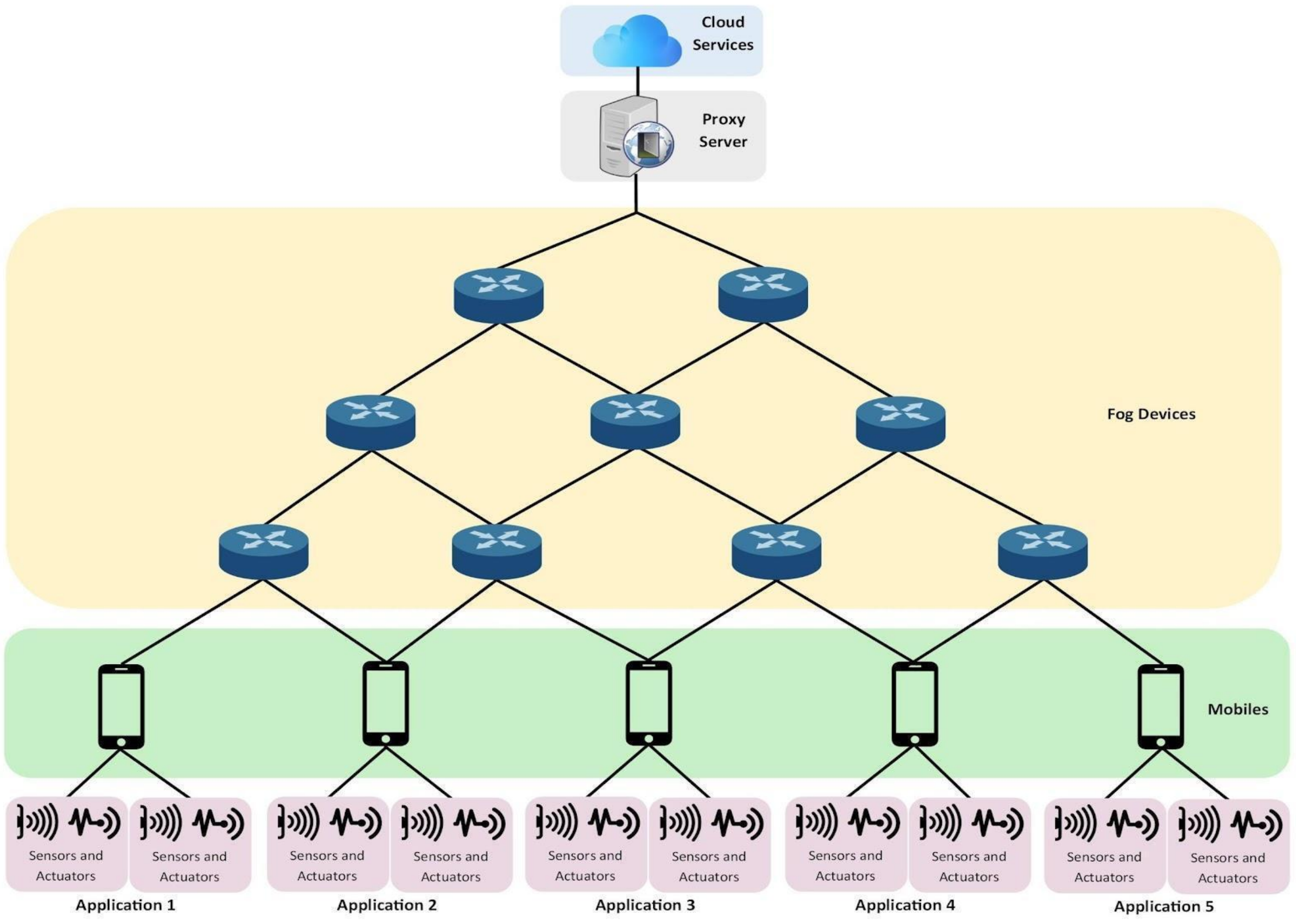


Fig 5.3: System topology of testing environment

Tree system topology in the testing environment is being shown in Figure 5.3. The model of the fog-cloud environment consists of several components such as the applications, mobiles, fog devices, proxy server and the cloud service. The reason for setting up the proxy server is to act as a gateway between the cloud and the fog devices.

Data Collection Method

Proposed QoE-Aware application allocation, QoE-Aware and Edgeward algorithm has been run several times of testing using the iFogSim simulation. The purpose of the testing is to obtain the data and results between the Proposed QoE-Aware application allocation, QoE-Aware and Edgeward algorithm. Finally, the QoE-Aware, Energy-Aware, and enhanced Offloading algorithm will be put together into a test and compared against the Edgeward algorithm.

The fog devices and modules are arranged in a strategic manner where the arrangement is based on the processing power of each of the fog devices and the processing power requirement for the modules.

The parameters obtained and collected are the execution time, energy consumption, and usage of the network for the purpose to compare the performance of the scheduling algorithm application module. Last but not least, total execution time of the application, energy consumption of the fog devices and the usage of the network are being used as performance metrics for comparison purposes.

Statistical Model

The indicator used to measure the accuracy of the estimate is the confidence interval. Next, the function of the confidence interval is to provide an accurate calculation on how close the measurement is to the initial estimation. The equation used to calculate can be referred below.

$$\underline{x} = \frac{\Sigma \quad x}{n} \text{ Calculate Mean Formula} \tag{5.1}$$

Eg: $\underline{x}$ *= (587 + 30152 + 32583 + 1152116 + 1978420 + 763 + 38278 + 424569 + 1214947 + 2458161 + 698 + 31568 + 329580 + 1190417 2305892) / 15 =* ***11188731 / 15 = 745,915.4***

The sum of all values obtained from the results and then divided by the total number of experiments n comes the mean of the data. Besides that, confidence intervals can be used to calculate standard deviation through the following equation.

$S = \sqrt{\frac{\Sigma \quad (x-\underline{x})^2}{n-1}}$ Calculate Sample Standard deviation Formula				(5.2)
x	$\underline{x}$	x-$\underline{x}$	$(x - \underline{x})^2$	
587	*745,915.4*	*-745,328.40*	*555514423846.56*	
30152	*745,915.4*	*-715,763.40*	*512317244779.56*	
32583	*745,915.4*	*-713,332.40*	*508843112889.76*	
1152116	*745,915.4*	*406,200.60*	*164998927440.36*	
1978420	*745,915.4*	*1,232,504.60*	*1519067589021.16*	
763	*745,915.4*	*-745,152.40*	*555252099225.76*	
38278	*745,915.4*	*-707,637.40*	*500750689878.76*	
424569	*745,915.4*	*-321,346.40*	*103263508792.96*	
1214947	*745,915.4*	*469,031.60*	*219990641798.56*	
2458161	*745,915.4*	*1,712,245.60*	*2931784994719.36*	
698	*745,915.4*	*-745,217.40*	*555348973262.76*	
31568	*745,915.4*	*-714,347.40*	*510292207886.76*	
329580	*745,915.4*	*-416,335.40*	*173335165293.16*	
1190417	*745,915.4*	*444,501.60*	*197581672402.56*	
2305892	*745,915.4*	*1,559,976.60*	*2433526992547.56*	
$\Sigma \quad (x - \underline{x})^2$ *= Total =*			*11441868243785.60*	

Eg: S = Sqrt (Total / (n - 1))

= Sqrt (11441868243785.60 / (15 - 1))

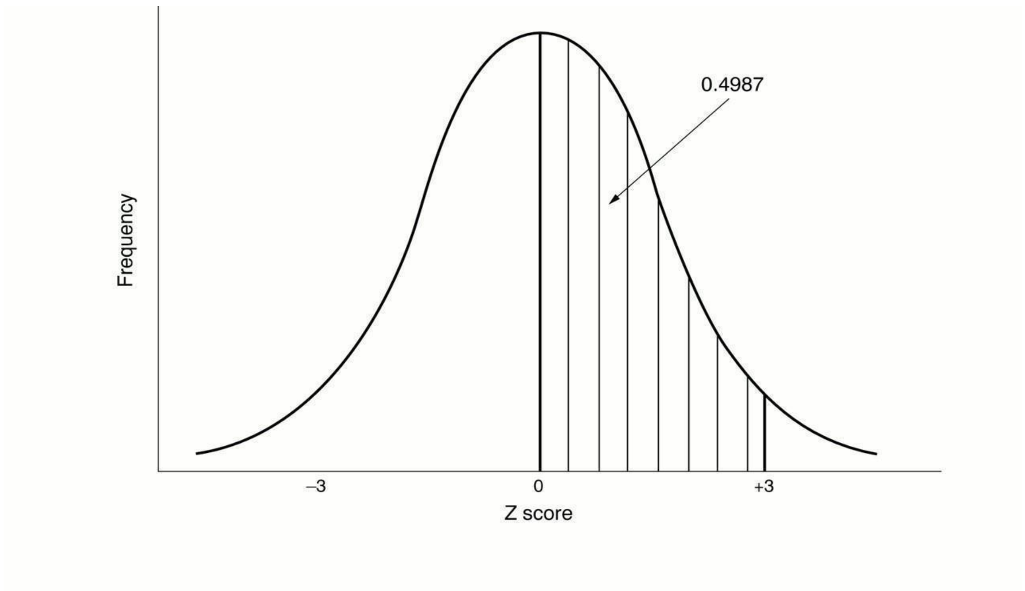
= 904033.352884473

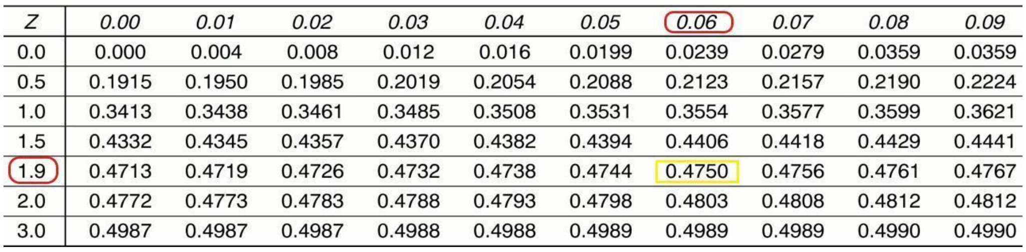
There are three values that are commonly used for confidence levels such as 90%, 95% and 99%. A decision has been made where 95% are chosen as the desired confidence level. Then, the calculation of Margin of Error can be calculated based on the following equation.

$$Margin\ of\ Error = Z_{\frac{\alpha}{2}} \times \frac{S}{\sqrt{(n)}} \tag{5.3}$$

Table 5.2: Notation for confidence interval

$\underline{x}$	Mean
n	Sample size
σ	Population Standard deviation
S	Sample Standard deviation
α	Confidence level
Z	Value refer to z table
$Z_{\frac{\alpha}{2}} \times \frac{\sigma}{\sqrt{(n)}}$	Margin Error
τ	T value





Z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.0	0.000	0.004	0.008	0.012	0.016	0.0199	0.0239	0.0279	0.0359	0.0359
0.5	0.1915	0.1950	0.1985	0.2019	0.2054	0.2088	0.2123	0.2157	0.2190	0.2224
1.0	0.3413	0.3438	0.3461	0.3485	0.3508	0.3531	0.3554	0.3577	0.3599	0.3621
1.5	0.4332	0.4345	0.4357	0.4370	0.4382	0.4394	0.4406	0.4418	0.4429	0.4441
1.9	0.4713	0.4719	0.4726	0.4732	0.4738	0.4744	0.4750	0.4756	0.4761	0.4767
2.0	0.4772	0.4773	0.4783	0.4788	0.4793	0.4798	0.4803	0.4808	0.4812	0.4812
3.0	0.4987	0.4987	0.4987	0.4988	0.4988	0.4989	0.4989	0.4989	0.4990	0.4990

Fig 5.4: Z-table

Z_(α/2) stands for confidence coefficient. At the beginning, the confidence level α is 95%. Next, division of the confidence level α value by 2 and obtain a value of 0.475. Hence, 0.475 is between the intersection of column 0.06 and row 1.9 based on the Z-table in Figure 5.4 and it shows that the critical value is 1.9 + 0.06 = 1.96. Next, the following equation can be used to define the confidence interval.

$$CI = \underline{x} \pm Z_{\frac{\alpha}{2}} \times \frac{S}{\sqrt{(n)}} \tag{5.4}$$

*Eg: CI = 745,915.4 $\pm$1.96 * (904033.352884473 / Sqrt(15))*

*= 745,915.4 $\pm$ 1.96 * 233420.408*

= 745,915.4 $\pm$ 457503.9997

= (288411.4003 , 1203419.3997)

We also will use paired sample T-test in order to make sure that QoE-Aware algorithm and QoE-Aware with Energy-Aware algorithm have significant differences with Edgeward algorithm. The equation 5.5 is used to calculate the T value and P value.

$$\tau = \frac{\left|\underline{x_1} - \underline{x_2}\right|}{\sqrt{\frac{\sigma^2}{n_1} + \frac{\sigma^2}{n_2}}} \qquad (5.5)$$

Eg: τ = |498132.13 - 608106.93| / Sqrt(489458.2852^2 */1+591935.3413*^{2}*/15)*

= |-109974.8| / Sqrt (15971294198 + 23359163219)

= |-109974.8| / Sqrt (39330457417)

= |-109974.8| / 198319.0798

= 0.5545

After the calculation and computation of previous equations, the values and information obtained will be recorded in tables in Section 5.4.

Performance Analysis

First and foremost, the results of the QoE-Aware algorithm such as the time execution and total network usage are obtained and recorded from the simulation and then further compared with the Edgeward algorithm. Next, the results for Energy-Aware with QoE-Aware from the simulation are recorded. In short, the QoE with Energy-Aware algorithm is compared with the Edgeward algorithm. Finally, the QoE-Aware, Energy-Aware, and enhanced Offloading algorithm will be tested and compared with the Edgeward algorithm.

In order to test the two algorithms which are the QoE-Aware with the Energy-Aware algorithm, we used several testing scenarios in deploying the application modules to the fog layer. Below is the table 5.3 that shows how the application modules are arranged in each test scenario.

Table 5.3 Test Scenario with their respective Application Module Arrangement

Scenario	Application Module Arrangement
Scenario 1	Module’s MIPS requirements increase from client towards last module.
Scenario 2	Module’s MIPS requirements decrease from client towards last module.
Scenario 3	Module’s MIPS requirements are in random order between client and last module.

Table 5.4 Module’s MIPS used in each scenario

Scenario	first module	Second module	Third module	Fourth module
1	100	200	300	400
2	600	500	400	300
3	random (100-1100)	random (100-1100)	random (100-1100)	random (100-1100)

In this testing, each application has 4 modules, and its MIPS will be used in accordance with Table 5.4. The fog devices and modules are arranged solely based on its processing power requirement. The nearer the fog devices to the cloud, the higher the processing power. In the first scenario, the modules’ MIPS are increased correspondingly, and the modules are placed in the ascending order of modules’ MIPS from end users towards the cloud. On the contrary, the modules' MIPS are decreased correspondingly, and the modules are placed in the descending order of modules’ MIPS from end users towards the cloud. However, the order of placing the modules and the modules’ MIPS are random in scenario 3. The main purpose of the test scenario is to evaluate and test the proposed mechanism in those three different scenarios. Therefore, the total number of three scenarios are formed and tested using the two algorithms.

Data Collection for performing QoE-Aware Algorithm

In this testing, each application has 4 modules, and its MIPS will be used in accordance with Table 5.4. The fog devices and modules are arranged solely based on its processing power requirement. The nearer the fog devices to the cloud, the higher the processing power. In the first scenario, the modules' MIPS are increased correspondingly, and the modules are placed in the ascending order of modules' MIPS from end users towards the cloud. On the contrary, the modules' MIPS are decreased correspondingly, and the modules are placed in the descending order of modules' MIPS from end users towards the cloud. However, the order of placing the modules and the modules' MIPS are random in scenario 3. The main purpose of the test scenario is to evaluate and test the proposed mechanism in those three different scenarios. Therefore, the total number of three scenarios are formed and tested using the two algorithms.

Table 5.5 shows the result of the Execution time of the application with the QoE-Aware algorithm and Edgeward tested.

Table 5.5: Execution time for Edgeward and QoE-aware algorithm

No of apps	**Scenario**	**QoE-Aware algorithm (ms)**	**Edgeward (ms)**
App 1	Scenario 1	1001	541
	Scenario 2	936	493
	Scenario 3	1350	535
App 2	Scenario 1	55805	77412
	Scenario 2	56838	78089
	Scenario 3	56396	75387
App 3	Scenario 1	658386	719644
	Scenario 2	635081	738338
	Scenario 3	671802	716679
App 4	Scenario 1	1124034	1254331
	Scenario 2	1213788	1258754
	Scenario 3	1203488	1234770
App 5	Scenario 1	862431	1685794
	Scenario 2	825641	1364587
	Scenario 3	856733	1685466
Total		8223710	10890820
Mean		548247.3333	726054.6667
SSD (Sample Standard Deviation)		473105.1415	649964.9823
CI (Margin of Error)		239424.2356	328927.6641
T Test		0.3989	
P Value		0.0350	

The table 5.5 clearly shows the algorithm with better execution time is the QoE-Aware algorithm in comparison with Edgeward algorithm. Next, the mean value is recorded and obtained where the QoE-Aware algorithm has the mean value of 548247.333 while the Edgeward algorithm has the mean value of 561727.2. Hence, the mean value result indicates that QoE-Aware has lower value which means it performs better than the Edgeward algorithm. Validation on the results are performed where T-test and P-value is calculated and computed. Furthermore, a value of 0.9458 is obtained for the T-test value and it means that the two values do not have significant difference while the P-value of 0.4585 is obtained for which it is greater than 0.05. The QoE-aware algorithm is faster than the Edgeward algorithm for 2.40%.

Table 5.6: Execution time for QoE-aware algorithm and proposed QoE-Aware Application Allocation

No of apps	**Scenario**	**QoE-Aware algorithm (ms)**	**QoE-Aware Application Allocation (ms)**
App 1	Scenario 1	1001	944
	Scenario 2	936	1178
	Scenario 3	1350	855
App 2	Scenario 1	55805	59530
	Scenario 2	56838	56161
	Scenario 3	56396	55052
App 3	Scenario 1	658386	722175
	Scenario 2	635081	740840
	Scenario 3	671802	758792
App 4	Scenario 1	1124034	441877
	Scenario 2	1213788	444978
	Scenario 3	1203488	336939
App 5	Scenario 1	862431	576487
	Scenario 2	825641	566645
	Scenario 3	856733	554598
Total		8223710	5317051
Mean		548247.3333	354470.0667
SSD (Sample Standard Deviation)		473105.1415	297481.5092
CI (Margin of Error)		239424.2356	150546.4150
T Test		0.1901	
P Value		0.0383	

The table 5.6 clearly shows the algorithm with better execution time is the QoE-Aware Application Allocation in comparison with QoE-Aware algorithm. Next, the mean value is recorded and obtained where the QoE-Aware Application Allocation has the mean value of 354470.0667 while the QoE-Aware algorithm has the mean value of 548247.333. Hence, the mean value result indicates that QoE-Aware Application Allocation has lower value which means it performs better than the previous QoE-Aware algorithm. Validation on the results are performed where T-test and P-value is calculated and computed. Furthermore, a value of 0.1901 is obtained for the T-test value and it means that the two values do have statistically significant difference while the P-value of 0.0191 is obtained for which it is smaller than 0.05. The QoE-Aware Application Allocation is faster than the QoE-Aware algorithm for 35.34%.

Table 5.7: Total energy consumption for QoE-aware algorithm and Edgeward algorithm

No of apps	**Scenario**	**QoE-Aware algorithm (mj)**	**Edgeward (mj)**
App 1	Scenario 1	7746133	7584735
	Scenario 2	7746133	7584101
	Scenario 3	7746133	7577828
App 2	Scenario 1	7818652	7776270
	Scenario 2	7819135	7782798
	Scenario 3	7817990	7776927
App 3	Scenario 1	7845183	7907466
	Scenario 2	7854322	7918060
	Scenario 3	7856753	7906792
App 4	Scenario 1	8036149	7936682
	Scenario 2	8071645	7894012
	Scenario 3	8036254	7914793
App 5	Scenario 1	8036273	7974861

	Scenario 2	8020127	7915943
	Scenario 3	8036154	7963174
Total		118487036	117414442
Mean		7899135.7333	7827629.4667
SSD (Sample Standard Deviation)		124200.4070	141544.1795
CI (Margin of Error)		62854.0781	71631.2380
T Test		0.1525	
P Value		0.00235	

Other than execution time, the next test will be testing the total consumption of energy for both QoE-Aware algorithm and Edgeward algorithm and the final result is being shown in table 5.7. As usual, there are a total number of 15 experiments performed that are accompanied by three different scenarios. Based on the table, the mean value for both algorithms have obtained and recorded which is 7899135.7333 for QoE-Aware algorithm and 7827629.4667 for Edgeward alone and the difference in the values indicates that the algorithm with the minimal or better at energy consumption is the Edgeward. Lastly, T-test and P value is calculated and obtained in order to validate the results where the T-test value shows the value of 0.1525 and P value shows the value of 0.00235 that was a significant difference between the Edgeward and QoE-Aware algorithm which is less than 0.05 because their total energy consumption readings are very close to each other.

Table 5.8: Total Energy consumption for QoE-aware algorithm and proposed QoE-Aware Application Allocation

No of apps	Scenario	QoE-Aware algorithm (mj)	QoE-Aware Application Allocation (mj)
App 1	Scenario 1	7746133	7746133
	Scenario 2	7746133	7746292
	Scenario 3	7746133	7747089
App 2	Scenario 1	7818652	7585916
	Scenario 2	7819135	7585878
	Scenario 3	7817990	7588085
App 3	Scenario 1	7845183	7884869
	Scenario 2	7854322	7858449
	Scenario 3	7856753	7883155
App 4	Scenario 1	8036149	7846451
	Scenario 2	8071645	7842035
	Scenario 3	8036254	7783430
App 5	Scenario 1	8036273	7881800
	Scenario 2	8020127	7878346
	Scenario 3	8036154	7883676
Total		118487036	116741604
Mean		7899135.7333	7782773.6
SSD (Sample Standard Deviation)		124200.4070	114107.3111
CI (Margin of Error)		62854.0781	57746.2668
T Test		0.0124	
P Value		0.0007	

Other than execution time, the next test will be testing the total consumption of energy for both QoE-Aware algorithm and proposed QoE-Aware Application Allocation and the final result is being shown in table 5.8. As usual, there are a total number of 15 experiments performed that are accompanied by three different scenarios. Based on the table, the mean value for both algorithms have obtained and recorded which is 7899135.7333 for QoE-Aware algorithm and 7782773.6 for QoE-Aware Application Allocation and the difference in the values indicates that the algorithm with the minimal or better at energy consumption is the QoE-Aware Application Allocation. Lastly, T-test and P value is calculated and obtained in order to validate the results where the T-test value shows the value of 0.0124 and P value shows the value of 0.0007 that was a significant difference between the for QoE-Aware Application Allocation and QoE-Aware algorithm which is less than 0.5. As a nutshell, the QoE-Aware Application Allocation has a 1.47% improvement compared to previous QoE-aware algorithm.

Table 5.9 shows the result of the Total Network Usage of the application with the QoE-Aware algorithm and Edgeward tested.

Table 5.9: Total Network Usage for Edgeward and QoE-aware algorithm

No of apps	**Scenario**	**QoE-Aware algorithm (kb)**	**Edgeward (kb)**
App 1	Scenario 1	14927.8	58963
	Scenario 2	14775.6	58983.4
	Scenario 3	14893	59057.8
App 2	Scenario 1	54263.8	86524.4
	Scenario 2	53079.8	87207
	Scenario 3	53409.2	85800.8
App 3	Scenario 1	47488	63762.2
	Scenario 2	47558.8	64187.4
	Scenario 3	47613.6	63394.8
App 4	Scenario 1	42967.3	53735.6
	Scenario 2	43625.1	53750.8
	Scenario 3	42381.2	53935.8
App 5	Scenario 1	41368.7	53648.2
	Scenario 2	41253.2	53104
	Scenario 3	41527	53973.7
Total		601132.1	950028.9
Mean		40075.4733	63335.26
SSD (Sample Standard Deviation)		13768.6751	12615.4170
CI (Margin of Error)		6967.9110	6384.2819
T Test		< 0.0001	
P Value		< 0.00001	

Table 5.9 shows the results of total network usage of QoE-Aware algorithm and Edgeward for a total of 15 experiments with three different scenarios. The results show that the QoE-Aware algorithm had a better total network usage compared to Edgeward. The mean of total network usage for QoE-Aware algorithm is 40075.4733 and for Edgeward is 63335.26. It is also observed that the total network usage for the QoE-Aware algorithm is lower than Edgeward. For validating the results, a T-test is used and the P value is calculated. The T-test shows < 0.0001 that indicates statistical significant difference between the two values, and it is found that the P value is < 0.00001 which is < 0.05. The QoE-aware algorithm will have lesser network usage for 36.72% while compared to Edgeward algorithm.

Table 5.10: Total Network Usage for QoE-Aware Application Allocation and QoE-aware algorithm

No of apps	**Scenario**	**QoE-Aware algorithm (kb)**	**QoE-Aware Application Allocation (kb)**
App 1	Scenario 1	14927.8	14836.6
	Scenario 2	14775.6	15031.4
	Scenario 3	14893	14896
App 2	Scenario 1	54263.8	52745
	Scenario 2	53079.8	52440.6
	Scenario 3	53409.2	52529.2
App 3	Scenario 1	47488	46089.2
	Scenario 2	47558.8	46094.6

	Scenario 3	47613.6	34267.8
App 4	Scenario 1	42967.3	38458.6
	Scenario 2	43625.1	38779.2
	Scenario 3	42381.2	30703.6
App 5	Scenario 1	41368.7	35471.6
	Scenario 2	41253.2	35630.4
	Scenario 3	41527	29696.8
Total		601132.1	537670.6
Mean		40075.4733	35844.7067
SSD (Sample Standard Deviation)		13768.6751	13215.6608
CI (Margin of Error)		6967.9110	6688.0471
T Test		0.3979	
P Value		0.0017	

Table 5.10 shows the results of total network usage of QoE-Aware algorithm and QoE-Aware Application Allocation for a total of 15 experiments with three different scenarios. The results show that the QoE-Aware Application Allocation had a better total network usage compared to QoE-Aware algorithm. The mean of total network usage for QoE-Aware algorithm is 40075.4733 and for QoE-Aware Application Allocation is 35844.7067. It is also observed that the total network usage for the QoE-Aware Application Allocation is lower than QoE-Aware algorithm. For validating the results, a T-test is used and the P value is calculated. The T-test shows 0.3979 that indicates significant difference between the two values, and it is found that the P value is 0.0017 which is < 0.05. QoE-Aware Application Allocation will have lesser network usage for 10.56% while compared to QoE-Aware algorithm.

Table 5.11: Rating Gain for QoE-Aware Application Allocation and QoE-aware algorithm

No. of apps	QoE-Aware Application Allocation (s)	QoE-Aware Algorithm (s)
1	35.30	
2	43.47	
3	43.47	
4	43.47	
5	43.40	
Total	209.11	
Mean	41.822	
SD		
CI		
T Test		
P Value		

The final results for resource gain for both algorithms have been recorded and obtained in table 5.11. The testing happens with 15 experiment counts. The scale can be explained in a way that as the mean value of the resource gain increases, the lower the cost induced. For instance, the QoE-Aware algorithm has the mean value of 1.95 which is greater than the Edgeward algorithm with the mean value of 1.396. Last but not least, validation of the result is performed by obtaining the T-test and P value. The T-test value shows the value of 3.2624 while the P value shows the value of 0.011486 that indicates a significant difference between the two values (QoE-Aware Application Allocation & Qoe-aware Algorithm) which is lower than 0.05.

Data Collection for performing Energy-aware Algorithm

The comparison of QoE-Aware with Energy-Aware algorithm and Edgeward happens in this section where execution time is the main focus. First of all, we put QoE-Aware with Energy-Aware algorithm into a test and three parameters are used for the comparison purpose which are the total usage of the network, total consumption of power and lastly the total execution time. Based on table 5.8, the table shows the final result between QoE-Aware with Energy-Aware algorithm and Edgeward algorithm in terms of execution time for a number of applications. Table 5.11 shows the result of the execution time for Edgeward, QoE-aware algorithm and QoE-aware with Energy-aware algorithm

Execution time for Edgeward, QoE-aware algorithm and QoE-aware with Energy-aware algorithm

No of apps	**Scenario**	**QoE-Aware algorithm (ms)**	**Edgeward (ms)**	**QoE-Aware with Energy-Aware algorithm (ms)**
App 1	Scenario 1	612	1100	410
	Scenario 2	597	1009	377
	Scenario 3	518	925	325
App 2	Scenario 1	25912	38170	17773
	Scenario 2	27600	39337	18147
	Scenario 3	26714	44569	18069
App 3	Scenario 1	601284	771059	218135
	Scenario 2	417536	455422	219815
	Scenario 3	312293	546385	222914
App 4	Scenario 1	1452502	869310	838680
	Scenario 2	1135091	870367	835160
	Scenario 3	1017145	886419	777496
App 5	Scenario 1	819166	1502511	821211
	Scenario 2	815274	1527898	819568
	Scenario 3	819738	1567123	818300
Total		7471982	9121604	5626380
Mean		498132.1333	608106.9333	375092
SSD (Sample Standard Deviation)		489458.2852	591935.3413	383333.5582
CI (Margin of Error)		247700.0682	299560.6139	193993.5463
T Test		0.2111550168		
P Value		0.0039895919		

After numerous counts of testing with different scenarios, it can be concluded that the algorithm with the best execution time than another is the algorithm of QoE-Aware with Energy-Aware compared to the Edgeward where the mean value for the combination of QoE-Aware with Energy-Aware algorithm is 375092.0, QoE-Aware algorithm is 498132.1 while Edgeward algorithm alone is 608106.9. Hence, the lower the mean value of an algorithm, the better the performance of the specific algorithm. In this case, the algorithm with better performance is the QoE-Aware algorithm with the Energy-Aware algorithm. Last but not least, the validation on the results has to be performed in order to test its accuracy by obtaining the values of T-test and P value. For instance, the T-test value and P value obtained are 0.2112 and 0.0039896 respectively. Furthermore, a value of 0.2112 is obtained for the T-test value and it means that the two values do not have significant difference while a value of 0.0039896 is obtained for which it is more than 0.05. However, QoE-Aware with the Energy-Aware algorithm still has 6.7834% faster than Edgeward.

Table 5.13: Total energy consumption for QoE-aware with Energy-aware algorithm and Edgeward algorithm

No of apps	**Scenario**	**QoE-Aware algorithm (mj)**	**Edgeward (mj)**	**QoE-Aware with Energy-Aware algorithm (mj)**
App 1	Scenario 1	7746133	7647869	7747072
	Scenario 2	7746292	7648675	7746486
	Scenario 3	7747089	7724308	7747192
App 2	Scenario 1	7817429	7775449	7889599
	Scenario 2	7818065	7780717	7916164
	Scenario 3	7820236	7838358	7877409
App 3	Scenario 1	7852352	7752522	8201050
	Scenario 2	7856059	7763205	8127141
	Scenario 3	7845164	7855144	8018881

App 4	Scenario 1	8038228	7849260	8227633
	Scenario 2	8002331	7831765	8114234
	Scenario 3	7980015	7861455	8423569
App 5	Scenario 1	8087449	7893617	8312432
	Scenario 2	8077500	7867069	8346382
	Scenario 3	8048200	7919988	8275659
Total		118482542	117009401	120970903
Mean		7898836.133	7800626.733	8064726.867
SSD (Sample Standard Deviation)		126024.9493	82692.6721	232740.9889
CI (Margin of Error)		63777.42388	41848.26601	117783.1913
T Test		0.000287444981		
P Value		0.00002361104868		

Other than execution time, the next test will be testing the total consumption of energy for both QoE-Aware with Energy-Aware algorithm and the Edgeward algorithm and the final result is being shown in table 5.12. As usual, there are a total number of 15 experiments performed that are accompanied by three different scenarios. Based on the table, the mean value for both algorithms have obtained and recorded which is 8064726.867 for QoE-Aware with Energy-Aware algorithm and 7800626.733 for Edgeward alone and the difference in the values indicates that the algorithm with the minimal or better at energy consumption is the QoE-Aware with Energy-Aware. Lastly, T-test and P value is calculated and obtained in order to validate the results where the T-test value shows the value of 0.00028744 and P value shows the value of 0.00002361 that indicates a significant difference between the Edgeward and QoE-Aware with Energy-Aware algorithm which is less than 0.05. Table 5.13 shows the result of the total network usage of the application with the QoE-Aware with Energy-Aware algorithm, QoE Algorithm and Edgeward tested.

Table 5.14: Total network usage for Edgeward, QoE-aware algorithm and QoE-aware with Energy-aware algorithm

No of apps	**Scenario**	**QoE-Aware algorithm (kb)**	**Edgeward (kb)**	**QoE-Aware with Energy-Aware algorithm (kb)**
App 1	Scenario 1	14964.0	58952.8	14851.8
	Scenario 2	14727.6	59014.4	14820.2
	Scenario 3	14985.6	58920.6	14821.4
App 2	Scenario 1	53648.2	86150.4	53269.2
	Scenario 2	53833.0	85887.4	53678.8
	Scenario 3	54187.6	86966.8	53923.4
App 3	Scenario 1	47649.2	63653.8	46725
	Scenario 2	47857.0	63374.8	47127.6
	Scenario 3	47389.0	64161.6	47308.2
App 4	Scenario 1	42884.0	54315.0	43056.6
	Scenario 2	43152.4	54050.6	42955.4
	Scenario 3	100898.8	54412.0	99285.4
App 5	Scenario 1	41952.2	50807.8	42116.2
	Scenario 2	41978.8	50928.6	41781
	Scenario 3	42131.0	51595.2	41937.8
Total		662238.4	943191.8	657658
Mean		44149.2	62879.45333	43843.86667
SSD (Sample Standard Deviation)		20918.06221	12924.2293	20579.9974
CI (Margin of Error)		10586.00006	6540.562448	10414.91566
T Test		0.005166914852		
P Value		0.002607611467		

The last parameter that is going to test is the network usage. Based on the table 5.14, the network usage of both algorithms are obtained and recorded through 15 times of experimentation accompanied by three different scenarios. The algorithm with the least network usage is the Edgeward with Energy-Aware algorithm where the mean value of it is 43843.86667 while the Edgeward algorithm alone has the mean value of 62879.45333. In conclusion, the lower the mean value of network usage, the less network usage of the algorithm. Last but not least, validation on the result is performed by obtaining the T-test and P value. The T-test value shows 0.0051669 while the P value shows 0.0026076 which is less than 0.05. It indicates a significant difference in the network usage between Edgeward and QoE-Aware with the Energy-Aware algorithm.

Data Collection for performing QoE-Aware Algorithm

The purpose of this section is to compare the algorithm between QoE-aware, Energy-aware placement with Computation Offloading and Edgeward algorithm alone. In summary, QoE-Aware with Energy-Aware algorithm is being combined with the Computation Offloading and compared against the Edgeward algorithm in order to obtain results of execution time, consumption of energy and lastly the usage of network. Table 5.15 shows the execution time for both algorithms.

Table 5.15: Execution time for Edgeward and QoE-aware with Energy-aware Placement with Enhanced Offloading algorithm

No of apps	**Scenario**	**QoE-aware with Energy-aware Placement with Offloading algorithm**	**Edgeward (ms)**
App 1	Scenario 1	519	1100
	Scenario 2	552	1009
	Scenario 3	544	925
App 2	Scenario 1	38345	38170
	Scenario 2	38120	39337
	Scenario 3	38529	44569
App 3	Scenario 1	410714	771059
	Scenario 2	412652	455422
	Scenario 3	415638	546385
App 4	Scenario 1	458394	869310
	Scenario 2	437355	870367
	Scenario 3	428320	886419
App 5	Scenario 1	656356	1502511
	Scenario 2	681074	1527898
	Scenario 3	667788	1567123
Total		4684900	9121604
Mean		312326.6667	608106.9333
SSD (Sample Standard Deviation)		264356.4314	591935.3413
CI (Margin of Error)		133782.8127	299560.6139
T Test		0.08811894506	
P Value		0.002418735049	

The results based on table 5.15 have been obtained and recorded for further analysis and observation. The testing happened with 15 experiment counts accompanied by three different scenarios. Based on the observation of the result obtained, the algorithm with better or efficient execution time is the Edgeward algorithm where it has the mean value of 312326.6667. Last but not least, validation on the results are performed where the T-test and P value is calculated and obtained. For instance, the T-test value shows 0.088118 and P value shows 0.0024187 which is less than 0.05. There is a significant difference between the Edgeward (ms) and QoE-aware and Energy-aware Placement with proposed offloading algorithm (ms) since P value less than 0.05 because their execution time's readings are very distinct to each other.

Table 5.16: Total energy consumption for Edgeward and QoE-aware with Energy-aware Placement with Enhanced Offloading Algorithm

No of apps	**Scenario**	**QoE-aware with Energy-aware Placement with Offloading algorithm**	**Edgeward (mj)**
App 1	Scenario 1	7746681	7647869
	Scenario 2	7747023	7648675
	Scenario 3	7747806	7724308
App 2	Scenario 1	7893655	7775449
	Scenario 2	7905772	7780717
	Scenario 3	7875590	7838358
App 3	Scenario 1	8177893	7752522
	Scenario 2	8185514	7763205
	Scenario 3	8010110	7855144
App 4	Scenario 1	8325028	7849260
	Scenario 2	8167156	7831765
	Scenario 3	8110226	7861455
App 5	Scenario 1	8447283	7893617
	Scenario 2	8494543	7867069
	Scenario 3	8668019	7919988
Total		121502299	117009401
Mean		8100153.267	7800626.733
SSD (Sample Standard Deviation)		290290.1853	82692.6721
CI (Margin of Error)		146907.103	41848.26601
T Test		0.0006385919742	
P Value		0.00009457846557	

Moreover, the next parameter that is going to be recorded and observed is the total consumption of energy for both comparison algorithms. The testing happens with 15 experiment counts accompanied by three different scenarios. In conclusion, the algorithm with least energy consumption is Edgeward algorithm with the mean value of 7882526.467 compared to QoE-Aware, Energy-Aware Placement with Computation Offloading Algorithm which has the mean value of 8100153.267. Last but not least, validation on the results are performed by obtaining the T-test and P value. For instance, the T-test shows 0.00063859 while the P value shows 0.000094578 which is less than 0.05 but their total energy consumption readings are very close to each other.

Table 5.17: Total network usage for Edgeward and QoE-aware with Energy-aware Placement with Enhanced Offloading Algorithm

No of apps	**Scenario**	**QoE-aware with Energy-aware Placement with Offloading algorithm**	**Edgeward (kb)**
App 1	Scenario 1	14934.8	58952.8
	Scenario 2	14905	59014.4
	Scenario 3	14920.2	58920.6
App 2	Scenario 1	53715.2	86150.4
	Scenario 2	53866.2	85887.4
	Scenario 3	54234.2	86966.8
App 3	Scenario 1	47372.8	63653.8
	Scenario 2	47380.2	63374.8
	Scenario 3	47278.4	64161.6
App 4	Scenario 1	42692.4	54315.0
	Scenario 2	43275.8	54050.6
	Scenario 3	43986.2	54412.0
App 5	Scenario 1	26019	50807.8
	Scenario 2	25813	50928.6
	Scenario 3	25875.2	51595.2
Total		556268.6	943191.8
Mean		37084.57333	62879.45333
SSD (Sample Standard Deviation)		14962.89276	12924.2293
CI (Margin of Error)		7572.268502	6540.562448
T Test		0.00002400455056	
P Value		0.0000004875142624	

Next, table 5.17 recorded the results of the total network usage for both algorithms in comparison. The testing happened with 15 experiment counts accompanied by 3 different scenarios. From the results obtained, it can be said that the algorithm with least usage of network is the Edgeward algorithm alone with the mean value of 62879.45333 compared to QoE-Aware, Energy-Aware Placement with proposed Offloading algorithm with the mean value of 37084.57333. Last but not least, validation on the results are performed by obtaining the T-test and P value. For instance, T-test has the value of 0.000024 and P value has the value of 0.00000048 which is less than 0.05 because their total network usage readings are very distinct to each other. QoE-aware, Energy-aware and offloading has lower network usage than edgeward. This is because fog devices are having a load limit which is used to avoid the fog devices being overloaded. When the modules exceed the fog device's load limit, the module will pass to another fog device. Therefore, the module would not only process solely on that fog device. Instead, the modules will pass to other fog devices for the sake of avoiding overload of fog devices. That's the reason why offloading ends up decreasing the network usage.

Chapter Summary And Evaluation

This chapter summarises the collection of data and using those data obtained to evaluate the proposed offloading algorithm with the comparison of existing QoE-aware algorithm, QoE-aware with energy-aware algorithm, and QoE-aware and energy-aware placement with computation offloading algorithm. The evaluation criteria including execution time, power consumption and network usage are used to compare between the proposed algorithm and existing algorithm. Other than that, the parameter of resource gain is used to compare the QoE-aware and Energy-aware Placement with the proposed Offloading Algorithm.

In short, the comparison of QoE-aware with proposed Offloading Algorithm against Edgeward algorithm and QoE-aware and Energy-aware Placement with proposed Offloading Algorithm has come to a conclusion where the proposed QoE-aware with proposed Offloading Algorithm has better a performance in terms of execution time, total network usage and resource gain than Edgeward algorithm.

RESULT AND DISCUSSION

This chapter compares and evaluates the performance of proposed algorithms including QoE, energy and offloading solutions. Execution Time, Total Energy consumption and Total Network Usage are used to evaluate the performance of the proposed algorithm. The purpose is to analyse the main objective of this research which is to improve data processing time and service based on QoE-Aware application mapping policy.

First, the data and results are collected from iFogSim simulation simulator. The result was analysed based on the data to show the impact of the proposed algorithm. In the experiments, the algorithm runs in the eclipse workspace. The three parameters include Execution Time, Total Energy consumption and Total Network Usage are captured. This chapter also focuses on the comparison of the results among Edgeward, QoE-aware algorithm, QoE-aware with energy-aware algorithm, QoE-aware with energy-aware and QoE, Energy-aware and offloading algorithm.

This chapter includes four sections. Section 6.1 presents the analysis of the result related to execution time; Section 6.2 presents the analysis of the result regarding total energy consumption; whereas Section 6.3 presents the analysis of result in terms of total network usage. Lastly, Section 6.4 concludes the chapter by highlighting the prominence of QoE-Aware offloading algorithm.

Analysis of Algorithms in term of Execution Time

Figure 6.1 shows the execution time of nine algorithms including Edgeward, QoE-aware algorithm, QoE-aware with energy-aware algorithm, QoE-aware with energy-aware and offloading algorithm, the energy-aware only, proposed QoE-aware application allocation, proposed QoE-aware with energy-aware algorithm, and proposed QoE-aware with energy-aware and offloading algorithm and the energy-aware only testing in proposed QoE-aware application allocation algorithm. On the y-axis represents the total execution time to complete the application's tasks whereas on the x-axis indicates different types of scenarios. The value for each of the algorithms is using the result of execution time of five applications.

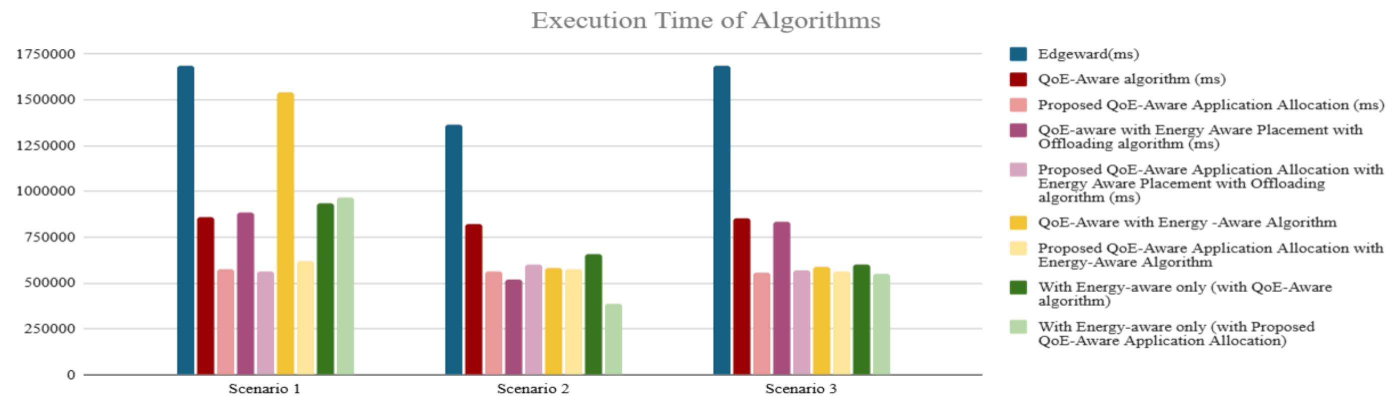


Figure 6.1: Execution time of algorithms in Scenarios

According to Figure 6.1, In scenario 1, 2 and 3 the QoE-Aware algorithm shows improved performance with reduced execution times compared to the Edgeward algorithm, but it still lags behind the more advanced approaches. The Proposed QoE-Aware Application Allocation achieves substantially lower execution times in all scenarios, demonstrating its effectiveness in optimizing task distribution. The QoE-Aware with Energy-Aware Placement with Offloading Algorithm further enhances performance by incorporating energy-efficient strategies, resulting in shorter execution times compared to the basic QoE-Aware algorithm. Among all algorithms, the Proposed QoE-Aware Application Allocation with Energy-Aware Placement and Offloading stands out, achieving the lowest execution times across all scenarios, making it the most efficient. The Energy-Aware Only algorithms, both with and without QoE-awareness, show moderate improvements over the Edgeward and basic QoE-Aware algorithms. However, their execution times are slightly higher compared to the combined strategies of QoE-awareness and energy-aware placement, highlighting the importance of integrating both strategies for optimal performance.

Analysis of Algorithms in term of Total Energy Consumption

Figure 6.2 shows the energy consumption of nine algorithms including Edgeward, QoE-aware algorithm, QoE-aware with energy-aware algorithm, QoE-aware with energy-aware and offloading algorithm, the energy-aware only, proposed QoE-aware application allocation, proposed QoE-aware with energy-aware algorithm, and proposed QoE-aware with energy-aware and offloading algorithm and the energy-aware only testing in proposed QoE-aware application allocation algorithm. On the y-axis represents the total energy consumption needed to complete the application's tasks whereas on the x-axis indicates different types of scenarios. The value for each of the algorithms is using the result of energy consumption of four applications. Based on the figure 6.2, QoE-aware with Energy-aware with offloading algorithm have the highest energy consumption whereas energy-aware algorithms use the lowest energy consumption.

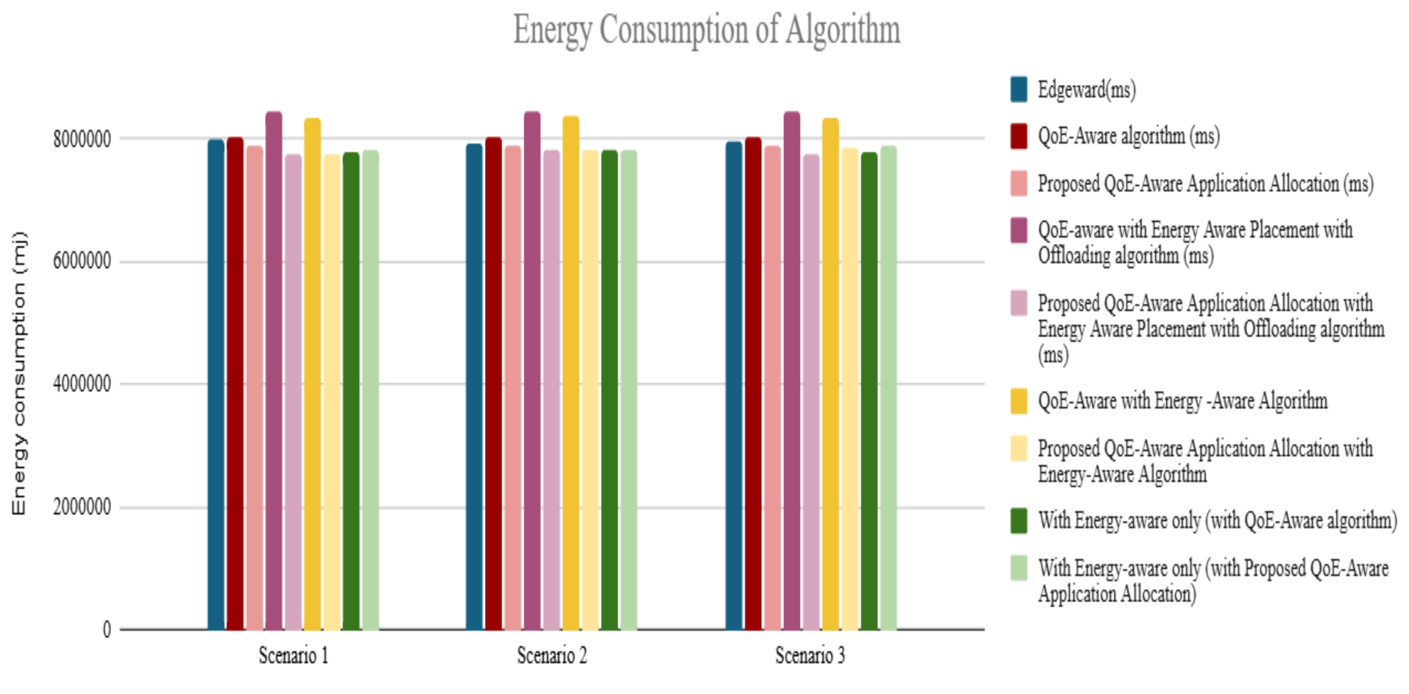


Figure 6.2: Total energy consumption of algorithms in Scenarios

Based on the scenarios, Edgeward has the fourth highest energy consumption among the others. It is caused by the lower layer of the fog devices which the closest to the end devices cannot process the task, it will pass to another that above them, the middle and upper layer fog devices. Therefore, the characteristics of Edgeward that transfer the modules from one to upper fog device will slightly increase the power consumption. Since the QoE, Energy-Aware and offloading algorithms have an extra process which is offloading newly arrived tasks so it has the highest energy consumption. The energy-aware only algorithm has the second lowest energy consumption because it executes the modules with barely sufficient energy so the energy usage of the fog devices that implement the energy-aware only algorithm will have the lowest energy consumption. The QoE-aware algorithm has energy consumption that is higher than the Edgeward because the QoE-aware algorithm has to fulfil the user requirements via many calculations which has increased the energy consumption. While for our proposed QoE-aware application allocation are having the least of energy consumption in all scenario.

Analysis of Algorithms in term of Total Network U

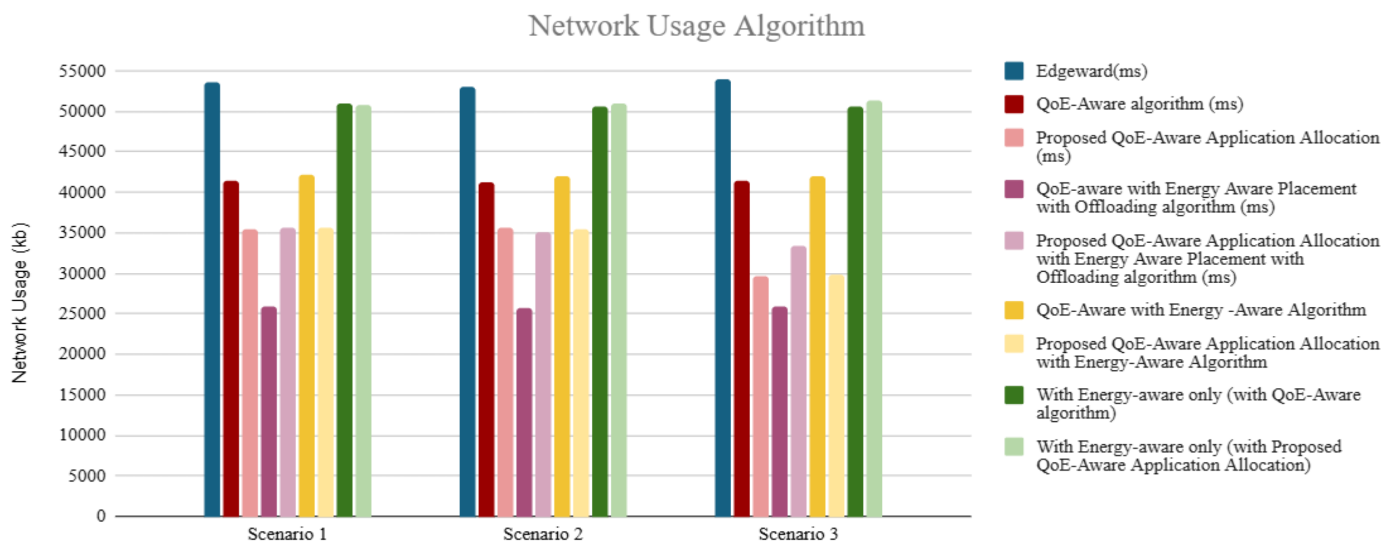


Figure 6.3: Total network usage of algorithms in Scenarios

Figure 6.3 shows the total network usage of nine algorithms including Edgeward, QoE-aware algorithm, QoE-aware with energy-aware algorithm, QoE-aware with energy-aware and offloading algorithm, the energy-aware only, proposed QoE-aware application allocation, proposed QoE-aware with energy-aware algorithm, and proposed QoE-aware with energy-aware and offloading algorithm and the energy-aware only testing in proposed QoE-aware application allocation algorithm. On the y-axis represents the total network usage needed to complete the application's tasks whereas on the x-axis indicates different types of scenarios. The value for each of the algorithms is using the result of network usage of five applications.

Based on the figure 6.3, the Edgeward algorithm consistently shows the highest network usage in all scenarios. The QoE-Aware algorithm and Proposed QoE-Aware Application Allocation demonstrate lower network usage, indicating better optimization of resources. Among all algorithms, the Proposed QoE-Aware Application Allocation with Energy-Aware Placement and Offloading achieves efficient network utilization, balancing task execution with minimal data transfer requirements. This efficiency makes it suitable for applications requiring reduced network overhead. The Proposed QoE-Aware Application Allocation optimizes network usage by dynamically distributing tasks based on Quality of Experience (QoE) metrics such as latency and bandwidth. This ensures effective data transmission and minimizes unnecessary network usage.

CONCLUSION

In this chapter, the experimental result is discussed to show the performance of the QoE, Energy-Aware and proposed algorithm on the basis of the execution time, total energy consumption and total network usage. The experiments were investigated based on comparison among algorithms such as Edgeward, QoE-aware algorithm, QoE-aware with energy-aware algorithm, QoE-aware with energy-aware and offloading algorithm , energy-aware only algorithm, proposed QoE-aware application allocation, proposed QoE-aware with energy-aware algorithm, and proposed QoE-aware with energy-aware and offloading algorithm and the energy-aware only testing in proposed QoE-aware application allocation algorithm.

In summary, the proposed QoE-aware application allocation performs better as compared to other algorithms. Moreover, the analysis of the results shows that the performance of the proposed algorithm needs lesser execution time and lesser total network usage than the existing QoE-aware with energy-aware and computation offloading algorithm.

REFERENCES

1. Abdali, T.-A. N., Hassan, R., Aman, A. H., & Nguyen, Q. N. (2021). Fog computing advancement: Concept, architecture, applications, advantages, and open issues. IEEE Access, 9, 75961–75980. https://doi.org/10.1109/access.2021.3081770
2. Abkenar, F.S., Ramezani, P., Iranmanesh, S., Murali, S., Chulerttiyawong, D., Wan, X., Jamalipour, A. and Raad, R., 2022. A Survey on Mobility of Edge Computing Networks in IoT: State-of-the-Art, Architectures, and Challenges. IEEE Communications Surveys & Tutorials.
3. Abofathi, Y., Anari, B., & Masdari, M. (2024). A learning automata based approach for module placement in Fog computing environment. Expert Systems with Applications, 237, 121607. https://doi.org/10.1016/j.eswa.2023.121607
4. Akyıldız, O., Kök, İ., Okay, F. Y., & Özdemir, S. (2023). A P4-assisted task offloading scheme for Fog networks: An intelligent transportation system scenario. Internet of Things, 22, 100695. https://doi.org/10.1016/j.iot.2023.100695
5. Amit Kumar Vishwakarma, Soni Chaurasia, Kumar, K., Yatindra Nath Singh, & Renu Chaurasia. (2024). Internet of things technology, research, and challenges: a survey. Multimedia Tools and Applications. https://doi.org/10.1007/s11042-024-19278-6
6. Avgeris, M, Spatharakis, D, Dechouniotis, D, Leivadeas, A, Karyotis, V & Papavassiliou, S 2022, ‘ENERDGE: Distributed Energy-Aware Resource Allocation at the Edge’, Sensors, vol. 22, no. 2.
7. Azizi, S, Shojafar, M, Abawajy, J & Buyya, R 2022, ‘Deadline-aware and energy-efficient IoT task scheduling in fog computing systems: A semi-greedy approach’, Journal of Network and Computer Applications, vol. 201.
8. Baranwal, G., Yadav, R. and Vidyarthi, D.P., 2020. QoE aware IoT application placement in fog computing using modified-topsis. Mobile Networks and Applications, 25(5), pp.1816-1832.
9. Bartosz Kopras, Bartosz Bossy, Idzikowski, F., Pawel Kryszkiewicz, & Bogucka, H. (2022). Task Allocation for Energy Optimization in Fog Computing Networks With Latency Constraints. IEEE Transactions on Communications, 70(12), 8229–8243. https://doi.org/10.1109/tcomm.2022.3216645
10. Bichi, BY, Islam, S ul, Kademi, AM & Ahmad, I 2022, ‘An energy-aware application module for the fog-based internet of military things’, Discover Internet of Things, vol. 2, no. 1.
11. Bikas, S., & Sayıt, M. (2024). Improving qoe with genetic algorithm-based path selection for MPTCP. IEEE Transactions on Network and Service Management, 1–1. https://doi.org/10.1109/tnsm.2024.3411104
12. Bridges, D., Pitiot, A., MacAskill, M.R. and Peirce, J.W., 2020. The timing mega-study: comparing a range of experiment generators, both lab-based and online. PeerJ, 8, p.e9414.
13. Carvalho, M., & Macedo, D. F. (2023). Container scheduling in co-located environments using Qoe Awareness. IEEE Transactions on Network and Service Management, 20(3), 3247–3260. https://doi.org/10.1109/tnsm.2023.3244090
14. Chang, Z., Liu, L., Guo, X., Chen, T. and Ristaniemi, T. 2020. Dynamic Resource Allocation and Computation Offloading for Edge Computing System. Artificial Intelligence Applications and Innovations. AIAI 2020 IFIP WG 12.5 International Workshops, 585(1868-422X), pp.61–73. doi:10.1007/978-3-030-49190-1_6
15. Charaf, L. A., Alihamidi, I., Deroussi, A., Saber, M., Ait Madi, A., & Addaim, A. (2021). Proposed access control architecture based on fog computing for IOT Environments. 2021 7th International Conference on Optimization and Applications (ICOA). https://doi.org/10.1109/icoa51614.2021.9442623
16. Chen, M., Xiao, Y., Li, Q. and Chen, K.C., 2020, June. Minimizing age-of-information for fog computing-supported vehicular networks with deep Q-learning. In ICC 2020-2020 IEEE International Conference on Communications (ICC) (pp. 1-6). IEEE.
17. Chen, X., Zhou, Y., Yang, L., & Lv, L. (2020). User satisfaction oriented resource allocation for fog computing: A mixed-task paradigm. IEEE Transactions on Communications, 68(10), 6470–6482. https://doi.org/10.1109/tcomm.2020.3008705
18. Cheng, Z., Gao, Z., Liwang, M., Huang, L., Du, X. and Guizani, M. 2021. Intelligent Task Offloading and Energy Allocation in the UAV-Aided Mobile Edge-Cloud Continuum. IEEE Network, [online] 35(5), pp.42–49. doi:10.1109/MNET.010.2100025.

19. Chuang, YT & Hsiang, CS 2022, ‘A popularity-aware and energy-efficient offloading mechanism in fog computing’, Journal of Supercomputing, vol. 78, no. 18, pp. 19435–19458.
20. Costa, B., Bachiega Jr, J., de Carvalho, L.R. and Araujo, A.P., 2022. Orchestration in fog computing: A comprehensive survey. ACM Computing Surveys (CSUR), 55(2), pp.1-34.
21. Delgado, C & Famaey, J 2022, ‘Optimal Energy-Aware Task Scheduling for Batteryless IoT Devices’, IEEE Transactions on Emerging Topics in Computing, vol. 10, no. 3, pp. 1374–1387.
22. Ding, Y., Liu, C., Zhou, X., Liu, Z. and Tang, Z. 2020. A Code-Oriented Partitioning Computation Offloading Strategy for Multiple Users and Multiple Mobile Edge Computing Servers. IEEE Transactions on Industrial Informatics, [online] 16(7), pp.4800–4810. doi:10.1109/TII.2019.2951206.
23. Evangeline, P., Chandrakasan, B., M, V. S., & Palanisamy, A. (2023b). A Fault Tolerant Multimedia Cloud Framework to Guarantee Quality of Experience (QoE) in Live Streaming. https://doi.org/10.21203/rs.3.rs-3144416/v1
24. Feng, J., Liu, L., Hou, X., Pei, Q., & Wu, C. (2023). QoE fairness resource allocation in digital twin-enabled wireless virtual reality systems. IEEE Journal on Selected Areas in Communications, 41(11), 3355-3367. https://doi.org/10.1109/JSAC.2023.3313195
25. Feng, W, Zhang, N, Lin, S, Li, S, Wang, Z, Ai, B & Zhong, Z 2022, ‘Energy-Efficient Collaborative Offloading in NOMA-Enabled Fog Computing for Internet of Things’, IEEE Internet of Things Journal, vol. 9, no. 15, pp. 13794–13807.
26. Fog Computing Market Size, Share | CAGR of 52.1%. (n.d.). Market.us. Retrieved June 21, 2024, from https://market.us/report/fog-computing-market/
27. Ghafari, R., & Mansouri, N. (2024). A novel energy-based task scheduling in fog computing environment: an improved artificial rabbits optimization approach. Cluster Computing. https://doi.org/10.1007/s10586-024-04396-5
28. Ghanavati, S, Abawajy, J & Izadi, D 2022, ‘An Energy Aware Task Scheduling Model Using Ant-Mating Optimization in Fog Computing Environment’, IEEE Transactions on Services Computing, vol. 15, no. 4, pp. 2007–2017.
29. Ghasemi, A. (2024). MOHHO: multi-objective Harris hawks optimization algorithm for service placement in fog computing. The Journal of Supercomputing. https://doi.org/10.1007/s11227-024-06389-y
30. Gupta, A., Bhadauria, H.S. and Singh, A., 2021. Load balancing based hyper heuristic algorithm for cloud task scheduling. Journal of Ambient Intelligence and Humanized Computing, 12(6), pp.5845-5852.
31. Hajam, S. S., & Sofi, S. A. (2023). Resource management in fog computing using greedy and semi-greedy spider monkey optimization. Soft Computing, 27(24), 18697–18707. https://doi.org/10.1007/s00500-023-09123-7
32. Hashemi, S. M., Sahafi, A., Rahmani, A. M., & Bohlouli, M. (2024). A new approach for service activation management in fog computing using Cat Swarm Optimization algorithm. Computing. https://doi.org/10.1007/s00607-024-01302-0
33. He, Z. and Peng, L., 2020. Evaluation Of Fog Topologies In Fog Planning For Iot Task Scheduling. New York: Association for Computing Machinery, pp.2177-2180.
34. Hossam, H. S., Abdel-Galil, H., & Belal, M. (2024). An energy-aware module placement strategy in fog-based healthcare monitoring systems. Cluster Computing. https://doi.org/10.1007/s10586-024-04308-7
35. Hosseinzadeh, M., Shankar, K., Apostolaki, M., Ramachandran, J., Adams, S. E., Sekar, V., & Sinopoli, B. (2023). CANE: A Cascade Control Approach for Network-Assisted Video QoE Management. IEEE Transactions on Control Systems Technology, 31(6), 2543-2553. https://doi.org/10.1109/TCST.2023.3267716
36. Hung, L.-H., Wu, C.-H., Tsai, C.-H., & Huang, H.-C. (2021). Migration-based load balance of virtual machine servers in cloud computing by load prediction using genetic-based methods. IEEE Access, 9, 49760–49773. https://doi.org/10.1109/access.2021.3065170
37. Idrees, A. K., Ali-Yahiya, T., Idrees, S. K., & Couturier, R. (2024). EDaTAD: Energy-Aware Data Transmission Approach with Decision-Making for Fog Computing-Based IoT Applications. Journal of Network and Systems Management, 32(3). https://doi.org/10.1007/s10922-024-09828-6

38. Idrees, AK, Ali-Yahiya, T, Idrees, SK & Couturier, R 2022, 'Energy-efficient fog computing-enabled data transmission protocol in tactile internet-based applications', in Proceedings of the ACM Symposium on Applied Computing, Association for Computing Machinery, pp. 206–209.
39. Iftikhar, S., Ahmad, M. M. M., Tuli, S., Chowdhury, D., Xu, M., Gill, S. S., & Uhlig, S. (2023). HunterPlus: AI based energy-efficient task scheduling for cloud–fog computing environments. Internet of Things, 21, 100667. https://doi.org/10.1016/j.iot.2022.100667
40. Islam, S., Ahammed, M., Siddique, N. A., Roy, P., Razzaque, Md. A., Hassan, M. M., & Saleem, K. (2024). A hyper-heuristic approach for quality of Experience Aware Service placement scheme in 5G Mobile Edge Computing. IEEE Access, 12, 72746–72765. https://doi.org/10.1109/access.2024.3403721Malik, U.M., Javed, M.A., Zeadally, S. and ul Islam, S., 2021. Energy efficient fog computing for 6G enabled massive IoT: Recent trends and future opportunities. IEEE Internet of Things Journal.
41. Jain, V., & Kumar, B. (2023). QoS-Aware Task Offloading in Fog Environment Using Multi-agent Deep Reinforcement Learning. Journal of Network and Systems Management, 31(7). https://doi.org/10.1007/s10922-022-09696-y
42. Khan, S. A., Abdullah, M., Iqbal, W., & Butt, M. A. (2022). Efficient job placement using two-way offloading technique over fog-cloud architectures. Cluster Computing, 26(6), 3503–3521. https://doi.org/10.1007/s10586-022-03750-9
43. Khan, S., Shah, I. A., Aurangzeb, K., Ahmad, S., Khan, J. A., & Anwar, M. S. (2024). Energy efficient task scheduling using fault tolerance technique for IoT applications in FOG computing environment. IEEE Internet of Things Journal, 1. https://doi.org/10.1109/jiot.2024.3403003
44. Laghari, A. A., Zhang, X., Shaikh, Z. A., Khan, A., Estrela, V. V., & Izadi, S. (2023). A review on quality of experience (QoE) in cloud computing. Journal of Reliable Intelligent Environments. https://doi.org/10.1007/s40860-023-00210-y
45. Li, H., Zhang, X., Li, H., Duan, X., & Xu, C. (2024). SLA-based task offloading for energy consumption constrained workflows in fog computing. Future Generation Computer Systems, 156, 64–76. https://doi.org/10.1016/j.future.2024.03.013
46. Liu, C., Liu, K., Guo, S., Xie, R., Lee, V.C.S. and Son, S.H. 2020. Adaptive Offloading for Time-Critical Tasks in Heterogeneous Internet of Vehicles. IEEE Internet of Things Journal, 7(9), pp.7999–8011. doi:10.1109/jiot.2020.2997720.
47. Liu, W., Li, C., Zheng, A., Zheng, Z., Zhang, Z., & Xiao, Y. (2023). FOG Computing Resource-Scheduling Strategy in IoT based on Artificial Bee colony Algorithm. Electronics, 12(7), 1511. https://doi.org/10.3390/electronics12071511
48. Liu, W., Zhang, H., Ding, H., Yu, Z., & Yuan, D. (2024). Qoe-aware collaborative edge caching and computing for adaptive video streaming. IEEE Transactions on Wireless Communications, 23(6), 6453–6466. https://doi.org/10.1109/twc.2023.3331724
49. Lu, H., Gu, C., Luo, F., Ding, W., Zheng, S. and Shen, Y. 2020. Optimization of Task Offloading Strategy for Mobile Edge Computing Based on Multi-Agent Deep Reinforcement Learning. IEEE Access, [online] 8, pp.202573–202584. doi:10.1109/ACCESS.2020.3036416.
50. Luxner, T. (2023, May 18). Cloud computing stats: Flexera 2023 State of the cloud report. Flexera Blog. https://www.flexera.com/blog/cloud/cloud-computing-trends-flexera-2023-state-of-the-cloud-report/
51. Mahmud, R., Srirama, S. N., Ramamohanarao, K., & Buyya, R. 2020. Quality of Experience (QoE)-aware placement of applications in Fog computing environments. Journal of Parallel and Distributed Computing. doi:10.1016/j.jpdc.2018.03.004
52. Manzoor, A., Shah, M.A., Khattak, H.A., Din, I.U. and Khan, M.K., 2022. Multi-tier authentication schemes for fog computing: Architecture, security perspective, and challenges. International Journal of Communication Systems, 35(12), p.e4033.
53. Mazur, I., Rak, J. and Nowicki, K., 2021. Ensuring the QoE-Related Fairness to Reduce the User Abandonment Ratio. Sensors, 21(21), p.7050.
54. Mirzapour-Moshizi, M., & Sattari-Naeini, V. (2022). QOE aware Application Placement in FoG environment using SAW Game Theory Method. Research Square (Research Square). https://doi.org/10.21203/rs.3.rs-2133563/v1

55. Mohammadzadeh, A., Zarkesh, M. A., Shahmohamd, P. H., Akhavan, J., & Chhabra, A. (2023). Energy-aware workflow scheduling in fog computing using a hybrid chaotic algorithm. the Journal of Supercomputing/Journal of Supercomputing, 79(16), 18569–18604. https://doi.org/10.1007/s11227-023-05330-z
56. Mordacchini, M, Ferrucci, L, Carlini, E, Kavalionak, H, Coppola, M & Dazzi, P 2022, Energy and QoE aware Placement of Applications and Data at the Edge, accessed from <http://ceur-ws.org>.
57. Mostafa, N. (2020). A dynamic approach for consistency service in cloud and fog environment. 2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC). https://doi.org/10.1109/fmec49853.2020.9144792
58. Naha, R, Garg, S, Battula, SK, Amin, MB & Georgakopoulos, D 2022, 'Multiple linear regression-based energy-aware resource allocation in the Fog computing environment', Computer Networks, vol. 216.
59. Nashaat, H., Ahmed, E. and Rizk, R, 2020, IoT Application Placement Algorithm Based on Multi-Dimensional QoE Prioritization Model in Fog Computing Environment. IEEE Access, 8, pp.111253–111264. doi:10.1109/access.2020.3003249.
60. Nazari Bu-Ali, A, Sohrabi Bu-Ali, S, Mohammadi, R, Nasiri Bu-Ali, M & Mansoorizadeh Bu-Ali, M 2022, 'IETIF: Intelligent Energy-aware Task Scheduling Technique in IoT/Fog Networks', accessed from <https://doi.org/10.21203/rs.3.rs-1454775/v1>.
61. POTU, N., BHUKYA, S., JATOTH, C., & PARVATANENI, P. (2022a). Quality-aware energy efficient scheduling model for fog computing comprised IOT Network. Computers &amp; Electrical Engineering, 97, 107603. https://doi.org/10.1016/j.compeleceng.2021.107603
62. Rahimikhanghah, A., Tajkey, M., Rezazadeh, B., & Rahmani, A. M. (2021). Resource scheduling methods in cloud and fog computing environments: A systematic literature review. Cluster Computing, 25(2), 911–945. https://doi.org/10.1007/s10586-021-03467-1
63. Raza, M.R., Varol, A. and Varol, N., 2020, June. Cloud and fog computing: A survey of the concept and challenges. In 2020 8th International Symposium on Digital Forensics and Security (ISDFS) (pp. 1-6). IEEE.
64. Reddy, P. B., & Sudhakar, C. (2023). An osmotic approach-based dynamic deadline-aware task offloading in edge–fog–cloud computing environment. the Journal of Supercomputing/Journal of Supercomputing, 79(18), 20938–20960. https://doi.org/10.1007/s11227-023-05440-8
65. Saif, F. A., Latip, R., Hanapi, Z. M., Alrshah, M. A., & Kamarudin, S. (2023). Workload allocation toward Energy Consumption-Delay Trade-Off in Cloud-Fog computing using Multi-Objective NPSO Algorithm. IEEE Access, 11, 45393–45404. https://doi.org/10.1109/access.2023.3266822
66. Saovapakhiran, B., Naruephiphat, W., Charnsripinyo, C., Baydere, S. and Ozdemir, S., 2022. QoE-Driven IoT Architecture: A Comprehensive Review on System and Resource Management. IEEE Access.
67. Sellami, B, Hakiri, A, Yahia, B & Berthou, 2022, P Energy-Aware Task Scheduling and Offloading using Deep Reinforcement Learning in SDN-enabled IoT Network,.
68. Shaifali P. Malukani , C. K. Bhensdadia, 2021. Fog Computing Algorithms: A Survey and Research Opportunities, vol. 26, no. 2, pp. 139–149
69. Sheikh Sofla, M., Haghi Kashani, M., Mahdipour, E., & Faghih Mirzaee, R. (2021). Towards effective offloading mechanisms in fog computing. Multimedia Tools and Applications, 81(2), 1997–2042. https://doi.org/10.1007/s11042-021-11423-9
70. Shukla, P., & Pandey, S. (2024). MOTORS: multi-objective task offloading and resource scheduling algorithm for heterogeneous fog-cloud computing scenario. the Journal of Supercomputing/Journal of Supercomputing. https://doi.org/10.1007/s11227-024-06315-2
71. Singh, J., Singh, P., Hedabou, M., & Kumar, N. (2023). An efficient Machine Learning-Based resource allocation scheme for SDN-Enabled FOG computing environment. IEEE Transactions on Vehicular Technology, 72(6), 8004–8017. https://doi.org/10.1109/tvt.2023.3242585
72. Singh, N & Das, AK 2022, 'Energy-efficient fuzzy data offloading for IoMT', Computer Networks, vol. 213.
73. Sreenivasu Mirampalli, Satish Narayana Srirama, Rajeev Wankar, & Raghavendra Rao Chillarige. (2022). Hierarchical fuzzy-based Quality of Experience (QoE)-aware application placement in fog nodes. Software: Practice and Experience, 53(2), 263–282. https://doi.org/10.1002/spe.3147

74. Suaad Hadi Hassan Al-Taai, Huda Abbas Kanber, & Waleed Abood Mohammed al-Dulaimi. (2023). The Importance of Using the Internet of Things in Education. International Journal of Emerging Technologies in Learning (IJET), 18(01), 19–39. https://doi.org/10.3991/ijet.v18i01.35999
75. Sulimani, H., Sulimani, R., Ramezani, F., Naderpour, M., Huo, H., Jan, T., & Prasad, M. (2024). HybOff: a Hybrid Offloading approach to improve load balancing in fog environments. Journal of Cloud Computing, 13(1). https://doi.org/10.1186/s13677-024-00663-3
76. Sumona, S. T., Hasan, S. S., Tamzid, A. Y., Roy, P., Razzaque, M. A., & Mahmud, R. (2024). A Deep Q-Learning Framework for Enhanced QoE and Energy Optimization in Fog Computing. Proceedings of the 2024 20th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT). IEEE. https://doi.org/10.1109/DCOSS-IoT61029.2024.00104
77. Swarnakar, S., Banerjee, C., Basu, J., & Saha, D. (2023). A multi-agent-based VM migration for dynamic load balancing in Cloud computing cloud environment. International Journal of Cloud Applications and Computing, 13(1), 1–14. https://doi.org/10.4018/ijcac.320479
78. Tariq, H, Javed, MA, Alvi, AN, Hasanat, MHA, Khan, MB, Saudagar, AKJ & Alkhathami, M 2022, ‘AI-Enabled Energy-Efficient Fog Computing for Internet of Vehicles’, Journal of Sensors, vol. 2022.
79. Tu, Y., Chen, H., Yan, L. and Zhou, X. 2022. Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning for Efficient Edge Computing in IoT. Future Internet, 14(2), p.30. doi:10.3390/fi14020030.
80. Varshney, S., Sandhu, R. and Gupta, P.K., 2021, October. QoE-based Resource Management of Applications in the Fog Computing Environment using AHP Technique. In 2021 6th International Conference on Signal Processing, Computing and Control (ISPCC) (pp. 669-673). IEEE.
81. Wang, B., Wang, C., Huang, W., Song, Y. and Qin, X. 2020. A Survey and Taxonomy on Task Offloading for Edge-Cloud Computing. IEEE Access, [online] 8(2169-3536), pp.186080–186101. doi:10.1109/ACCESS.2020.3029649.
82. Wang, J., Hu, J., Min, G., Zhan, W., Zomaya, A.Y. and Georgalas, N. 2022. Dependent Task Offloading for Edge Computing based on Deep Reinforcement Learning. IEEE Transactions on Computers, [online] 71(10), pp.2449–2461. doi:10.1109/TC.2021.3131040.
83. Wang, L., Li, C., Dai, W., Li, S., Zou, J. and Xiong, H., 2022. QoE-Driven Adaptive Streaming for Point Clouds. IEEE Transactions on Multimedia.
84. Xu, J., Gu, B. and Tian, G., 2022. Review of agricultural IoT technology. Artificial Intelligence in Agriculture.
85. Yadav, R., & Baranwal, G. (2023). A study on integration of trust management and application placement in Fog Computing. 2023 International Conference on Electrical, Electronics, Communication and Computers (ELEXCOM). https://doi.org/10.1109/elexcom58812.2023.10370242
86. Yan, L., Chen, H., Tu, Y. and Zhou, X. 2022. A Task Offloading Algorithm With Cloud Edge Jointly Load Balance Optimization Based on Deep Reinforcement Learning for Unmanned Surface Vehicles. IEEE Access, [online] 10, pp.16566–16576. doi:10.1109/ACCESS.2022.3150406.
87. Yang, M., Zhu, H., Wang, H., Koucheryavy, Y., Samouylov, K. and Qian, H. 2021. An Online Learning Approach to Computation Offloading in Dynamic Fog Networks. IEEE Internet of Things Journal, [online] 8(3), pp.1572–1584. doi:10.1109/JIOT.2020.3015522.
88. Yang, Z., Zhang, Y., & Tian, J. (2022). The effect of QoS and QoE requirements for designing task processing controller based on fuzzy logic on IoT environments. Cluster Computing, 26(2), 1267–1283. https://doi.org/10.1007/s10586-022-03586-3
89. Zhao, H., Xu, J., Li, P., Feng, W., Xu, X., & Yao, Y. (2024). Energy minimization partial task offloading with joint dynamic voltage scaling and transmission power control in fog computing. IEEE Internet of Things Journal, 11(6), 9740–9751. https://doi.org/10.1109/jiot.2023.3324196
90. Zhao, T., He, L., Huang, X. and Li, F., 2021. QoE-Driven Secure Video Transmission in Cloud-Edge Collaborative Networks. IEEE Transactions on Vehicular Technology, 71(1), pp.681-696