JOB SCHEDULING ON FOG COMPUTING INFRASTRUCTURE BASED ON QOS PARAMETERS

Project report submitted in partial fulfillment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering

By

Achyut Tiwari (191391)

Under the supervision of

Dr. Rajni Mohana

to

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

CERTIFICATE

I hereby declare that the work presented in this report entitled "Job Scheduling on Fog Computing Infrastructure based on QoS Parameters" in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from January 2023 to May 2023 under the supervision of Dr Rajni Mohana, Associate Professor, Department of Computer Science and Engineering and Information Technology. The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Achyut Tiwari,
191391

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Dr Rajni Mohana
Associate Professor
Department of Computer Science & Engineering
and Information Technology
Dated:

PLAGIARISM CERTIFICATE

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: ....

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper

Name: ________________________ __**Department:** _______________ **Enrolment No** ________

Contact No. ___________________________**E-mail.** __________________________________

Name of the Supervisor: ____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): ______________________

____

____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at ....(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	**Excluded**	**Similarity Index (%)**	**Generated Plagiarism Report Details** (Title, Abstract & Chapters)	
	• All Preliminary Pages • Bibliography/Images/Quotes • 14 Words String		Word Counts	
Report Generated on			Character Counts	
		Submission ID	Total Pages Scanned	
			File Size	

Checked by
Name & Signature **Librarian**

....

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com

ACKNOWLEDGEMENT

First, I express my heartiest thanks and gratefulness to Lord Shiva for His divine blessing to make it possible to complete the project work successfully.

I am grateful and wish my profound indebtedness to Dr Rajni Mohana**,** Associate Professor, Department of CSE & IT, Jaypee University of Information Technology, Waknaghat. Deep Knowledge & keen interest of my supervisor in the field of "Cloud Computing" to carry out this project. Her endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, and reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I am deeply indebted to Professor Vivek Kumar Sehgal, Head, Department of CSE & IT at the Jaypee University of Information Technology for his constant encouragement and motivation that enthused me to complete my project with zest and determination.

I would like to express my sincere appreciation to Dr Aman Sharma, Assistant Professor in the Department of CSE and IT at the Jaypee University of Information Technology, for their insightful recommendations during the course of my projects.

I would also generously welcome my friend Aishani Pachauri and each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

No expression of appreciation is complete without recognition of the prayers, good wishes, advice and moral support of my affectionate parents Dr Rajesh Tiwari and Mrs Alka Tiwari, which helped me immensely to achieve my goal.

Name: Achyut Tiwari

Project Group No. 95

Roll No: 191391

TABLE OF CONTENT

LIST OF ABBREVIATIONS

1. WSG: Watt Strogatcz Generator
2. BAG: Barbasi Albert Generator
3. ER: Erdos- Renyi
4. EF: Edge Fog Computing
5. IoT: Internet of things
6. QoS: Quality of Service
7. GDPR: General Data Protection Regulation
8. MEC: Multi-Access Edge Computing
9. CC: Cloud Computing
10. FC: Fog Computing
11. AI: Artificial Intelligence
12. ML: Machine Learning

LIST OF FIGURES

LIST OF GRAPHS

LIST OF TABLES

ABSTRACT

The increasing demand for cloud computing in the Internet of Things (IoT) device ecosystem has highlighted the need for a more suitable infrastructure. Edge and Fog environments have emerged as viable options, as they allow for computation to be performed closer to the relevant resources, thereby reducing latency. The dynamic nature of IoT traffic and the demand for low latency services further underscore the importance of efficient service placement, which is facilitated by Fog computing through the implementation of load balancing. In this study, the authors focus on two critical parameters for IoT devices: load balancing and Quality of Service (QoS) improvement. To address these concerns, the authors propose a decentralized, agent-based solution that uses edge-to-cloud nodes to cooperatively manage the network's input load. The solution utilizes agents that generate potential request paths for resource allocation and choose the path that maximizes edge usage while minimizing service execution cost. The proposed solution is expected to effectively balance the input load throughout the network and reduce the cost-of-service execution in the IoT device ecosystem.

Chapter-1
INTRODUCTION

1.1 Introduction

New age system design is far away from traditional monolithic architecture, utilising the microservices concept which not only distributes functionality but takes advantage of shared resources [1]. This enables continuous delivery and deployment of large-scale applications. These system designs deployed on cloud environments are extended to Edge-Fog computing environments which bring resources near the computation and find their best applications in the field of Internet of things (IoT) based applications. These applications are often latency sensitive and require a scalable Edge-Fog infrastructure [2,3]. Growth in heavy applications all around the globe expects a delay-sensitive cloud computing paradigm. Keeping in mind the upcoming use of 5G technology, edge computing made its way to facilitate storage on the cloud for devices to function optimally. Given a certain pool of users, there are edge servers located in a given area to provide for such need for space and data allocation processes. This helps users in offloading the impending bulky data to the nearby edge server deployed in the area to which the user is mapped.

The IoT service placement problem is a problem that arises when trying to determine how to best distribute a service across a set of servers so that each server is not overloaded and the service as a whole can handle the load [4]. This problem is especially relevant in the context of the Internet of Things, where there may be a large number of devices making requests to the service all at same time [5,6,7].There are several different ways to approach this problem, but one common approach is to use a load-balancing algorithm. The proposed algorithm will take into account the number of requests that each server is receiving and then redistribute the load so that each server is handling a more equal number of requests. This can help to ensure that no single server is overloaded and that the service as a whole can keep up with the demand. Several different load-balancing algorithms can be used, and the best one to use will likely depend on the specific details of the IoT service and the servers that it is running on. However, some common algorithms that could be used include round-robin, least-loaded, and least-connections. No matter which algorithm is used, the goal is to distribute the load as evenly as possible across the servers so that each one can handle the workload without being overwhelmed. This can help to ensure that the service can keep up with demand and provide a good experience to users.

Load-balancing algorithms are used to distribute the load across servers, so that each server can handle a more equal number of requests. This helps to ensure that no single server is overloaded, and that the service as a whole can keep up with the demand. Several different load-balancing algorithms can be used, and the best one will likely depend on the specific details of the IoT service and the servers that it is running on. Round-robin, least-loaded, and least-connections are some of the common load-balancing algorithms that can be used.
To address the IoT service placement problem, there are different techniques, including heuristic and metaheuristic algorithms. Heuristic algorithms are designed to provide a feasible solution in a reasonable amount of time but do not guarantee an optimal solution. Metaheuristic algorithms, on the other hand, are designed to find near-optimal solutions and can handle complex optimization problems. There are several metaheuristic algorithms that have been applied to the IoT service placement problem, including genetic algorithms, simulated annealing, ant colony optimization, and particle swarm optimization.

As IoT devices become more prevalent, the amount of data generated by them increases exponentially. This data needs to be processed in real-time to enable near-instant decision-making. However, the traditional cloud computing paradigm does not provide the required low latency for such applications. This is where edge computing comes in, with its ability to bring resources closer to the data source, reducing latency and increasing processing speed.
Edge computing involves deploying servers closer to the data source, which can be a device or a sensor. These servers are located in proximity to the data source,

providing low latency and reducing the load on the cloud. Edge computing also provides the advantage of reduced network bandwidth requirements, which can be significant in IoT applications.
However, edge computing also presents its own set of challenges. For instance, it is difficult to determine how to distribute the IoT service across a set of servers so that each server is not overloaded and the service as a whole can handle the load. This problem is known as the IoT service placement problem and can be addressed using load-balancing algorithms.

One of the load-balancing algorithms that can be used is the round-robin algorithm. This algorithm distributes requests evenly across servers by sending each request to the next server in the sequence. The least-loaded algorithm distributes requests to the server with the least amount of load. The least-connections algorithm distributes requests to the server with the least number of connections. These algorithms ensure that each server is handling a more equal number of requests, reducing the likelihood of overloading any one server. In addition to load-balancing algorithms, metaheuristic algorithms such as genetic algorithms, simulated annealing, ant colony optimization, and particle swarm optimization can also be used to solve the IoT service placement problem. These algorithms are designed to find near-optimal solutions and can handle complex optimization problems. For instance, genetic algorithms mimic the process of natural selection to find an optimal solution. Simulated annealing simulates the annealing process of metals to find a solution that minimizes energy. Ant colony optimization mimics the behaviour of ants to find the shortest path between two points. Particle swarm optimization simulates the behaviour of particles to find the best solution.

To make edge computing more efficient, it is also important to consider the physical placement of servers. For instance, servers should be placed in proximity to the data source to reduce latency. The number of servers should also be determined based on the workload and the capacity of the servers. Moreover, the architecture of the system should be designed in a modular way to enable scalability and ease of maintenance.

Another consideration in edge computing is security. Since edge computing involves deploying servers closer to the data source, there is a risk of data breaches. It is, therefore, important to ensure that the servers are secure and that the communication between the servers and the cloud is encrypted. Moreover, access to the servers should be restricted to authorized personnel only.

In conclusion, edge computing is a promising technology for processing data generated by IoT devices. However, it presents its own set of challenges, including the IoT service placement problem. Load-balancing algorithms and metaheuristic algorithms can be used to solve this problem. Moreover, the physical placement of servers, the architecture of the system, and security considerations should also be taken into account to make edge computing more efficient and secure. As IoT applications continue to grow, edge computing will become even more important, and further research is needed to address the challenges it presents.

1.2 Problem Statement

IoT applications such as those used for smart homes, smart cities, and smart healthcare must be able to adapt to changing user behaviour, as users' actions may vary based on different factors. Additionally, many of these applications require timely responses. Consider a city that uses IoT sensors to monitor and manage its transportation system, including traffic flow, parking availability, and public transportation routes.

Internet of Things (IoT) is transforming many industries by enabling them to collect, process and analyse real-time data. One of the industries that IoT is having a significant impact on is the transportation industry. Smart cities are using IoT sensors to monitor traffic, parking availability, and public transportation to provide efficient transportation systems. However, providing high-quality service to the citizens requires the IoT sensors to ensure a high level of quality of service (QoS).

QoS is a measure of the overall performance of a system, including its ability to deliver data reliably, quickly, and with low latency. In the context of IoT applications, QoS is critical because many applications require real-time data to be delivered in a timely manner. For instance, in a smart city transportation system, real-time data on traffic flow, parking availability, and public transportation routes must be delivered quickly to enable efficient transportation systems.

To achieve high QoS, IoT sensors must use QoS-aware protocols and algorithms that prioritize the delivery of critical data over less critical data. Prioritizing the delivery of critical data ensures that real-time information is delivered to the transportation management system in a timely and reliable manner. For example, in a smart parking system, information on parking availability is critical, and it needs to be delivered in real-time. If the information is delayed or unreliable, it can cause congestion and increase pollution levels.

Moreover, IoT sensors must be able to adapt to changes in network conditions and traffic load to ensure optimal performance and QoS. For instance, in a smart transportation system, the amount of data that needs to be delivered varies depending on the time of day and the day of the week. During peak hours, there may be a higher demand for real-time data, and the network may be congested. In this case, the IoT sensors must be able to adapt to the changing network conditions and traffic load to ensure optimal performance and QoS.
Cost is another important parameter that needs to be considered when managing resources and allocating resources in IoT applications. In many cases, the cost of deploying and maintaining an IoT system can be significant. Therefore, it is essential to use cost-effective solutions that can deliver high QoS.

One approach to reducing the cost of IoT systems is to use edge computing. Edge computing enables data to be processed and analysed at the edge of the network, closer to the data source. By processing and analysing data at the edge of the network, the amount of data that needs to be transmitted over the network can be reduced, which can reduce the cost of the system. Moreover, edge computing can improve QoS by reducing latency and increasing reliability.

In conclusion, IoT is transforming many industries, including the transportation industry. Smart cities are using IoT sensors to monitor traffic, parking availability, and public transportation to provide efficient transportation systems. To achieve high QoS, IoT sensors must use QoS-aware protocols and algorithms that prioritize the delivery of critical data over less critical data. Moreover, IoT sensors must be able to adapt to changes in network conditions and traffic load to ensure optimal performance and QoS. Cost is another important parameter that needs to be considered when managing resources and allocating resources in IoT applications. Edge computing is a cost-effective solution that can improve QoS by reducing latency and increasing reliability.

To provide an efficient transportation system, the IoT sensors need to ensure a high level of QoS, such as low latency and high reliability, to deliver real-time information to the city's transportation management system. For example, if the sensors providing information on parking availability are slow or unreliable, drivers may waste time and fuel looking for parking spots, resulting in traffic congestion and increased air pollution. To address this problem, the IoT sensors can use QoS-aware protocols and algorithms that prioritize the delivery of critical data, such as parking availability and traffic flow, over less critical data, such as weather information. These protocols can also adapt to changes in network conditions and traffic load to ensure optimal performance and QoS.

Given the above application scenario, it is very vital to manage resources and allocate the resources in consideration of two vital parameters which are Cost and Quality of service.

1.3 Objectives

a) To propose a solution based on distributed agents for plan generation and selection for service placement on IOT devices.

b) To produce a load balancing technique in the Fog-Cloud environment which minimised cost related to service execution.

1.4 Proposed Methodology

In the rapidly evolving landscape of cloud-fog computing, the deployment of distributed agents offers a promising approach to enhance the scalability, reliability, performance, and security of complex and demanding applications. With the exponential growth of data and the increasing complexity of applications, traditional centralized systems have become inadequate in handling large workloads and providing efficient processing. Distributed agents, on the other hand, can be deployed across the cloud-fog environment to improve the overall efficiency of the system.

The scalability of the system can be improved by deploying distributed agents that can handle large workloads effectively. The distribution of agents across the cloud-fog environment ensures that the load is distributed evenly, preventing any single agent from becoming overloaded. This, in turn, enables the system to process a large number of requests simultaneously, improving its scalability. Furthermore, the deployment of distributed agents enables the system to continue operating even if one agent fails, enhancing reliability and reducing the risk of downtime. This approach also provides flexibility, as distributed agents can be designed to be highly adaptable to changes in workload or environmental conditions, enabling more efficient resource allocation and task delegation.

In addition to scalability and reliability, distributed agents can also enhance the performance of the system. By enabling faster and more efficient processing, distributed agents can improve the overall responsiveness of the system. The redundancy provided by the deployment of agents in multiple locations also improves the resilience of the system, ensuring that critical services remain available even if one part of the cloud-fog environment experiences a failure. This approach also enables the system to handle dynamic workloads more effectively, as distributed agents can be designed to adjust their processing capabilities in response to changing workload conditions.

Moreover, the deployment of distributed agents can enhance the security of the system by using encryption, authentication, and other security measures to prevent unauthorized access and protect sensitive data. By decentralizing the system, the risk of a single point of failure is reduced, making it more difficult for malicious actors to disrupt the system. Furthermore, distributed agents can be designed to operate in a more secure manner than traditional centralized systems, as they can leverage the latest security technologies and techniques to prevent attacks and breaches.

In summary, the deployment of distributed agents in cloud-fog environments offers numerous benefits, including scalability, reliability, performance, and security. These benefits make distributed agents well-suited for complex and demanding applications, as they can enhance the efficiency and adaptability of the system. By leveraging the advantages of distributed agents, organizations can build more robust and scalable cloud-fog environments that can adapt to changing workloads and environmental conditions, while maintaining high levels of performance and security. As the use of cloud-fog environments continues to grow, the deployment of distributed agents will become increasingly important in enabling organizations to achieve their goals and deliver high-quality services and applications to their customers.

In summary, the benefits of distributed agents in cloud-fog environments make them well-suited for complex and demanding applications. They can improve the efficiency, reliability, and security of the system, enabling it to better support a wide range of services and applications. By leveraging the advantages of distributed agents, organizations can build more robust and scalable cloud-fog environments that can adapt to changing workloads and environmental conditions, while maintaining high levels of performance and security.

As we see in the figure 1, A model for depiction of plan generation and selection for service placement is made using Distributed Agents.

Service placement is a critical function for managing the Internet of Things (IoT). By understanding the service requirements of devices and applications, and mapping these to the capabilities of available IoT platforms, service providers can ensure that devices and applications are able to interoperate and function as intended.

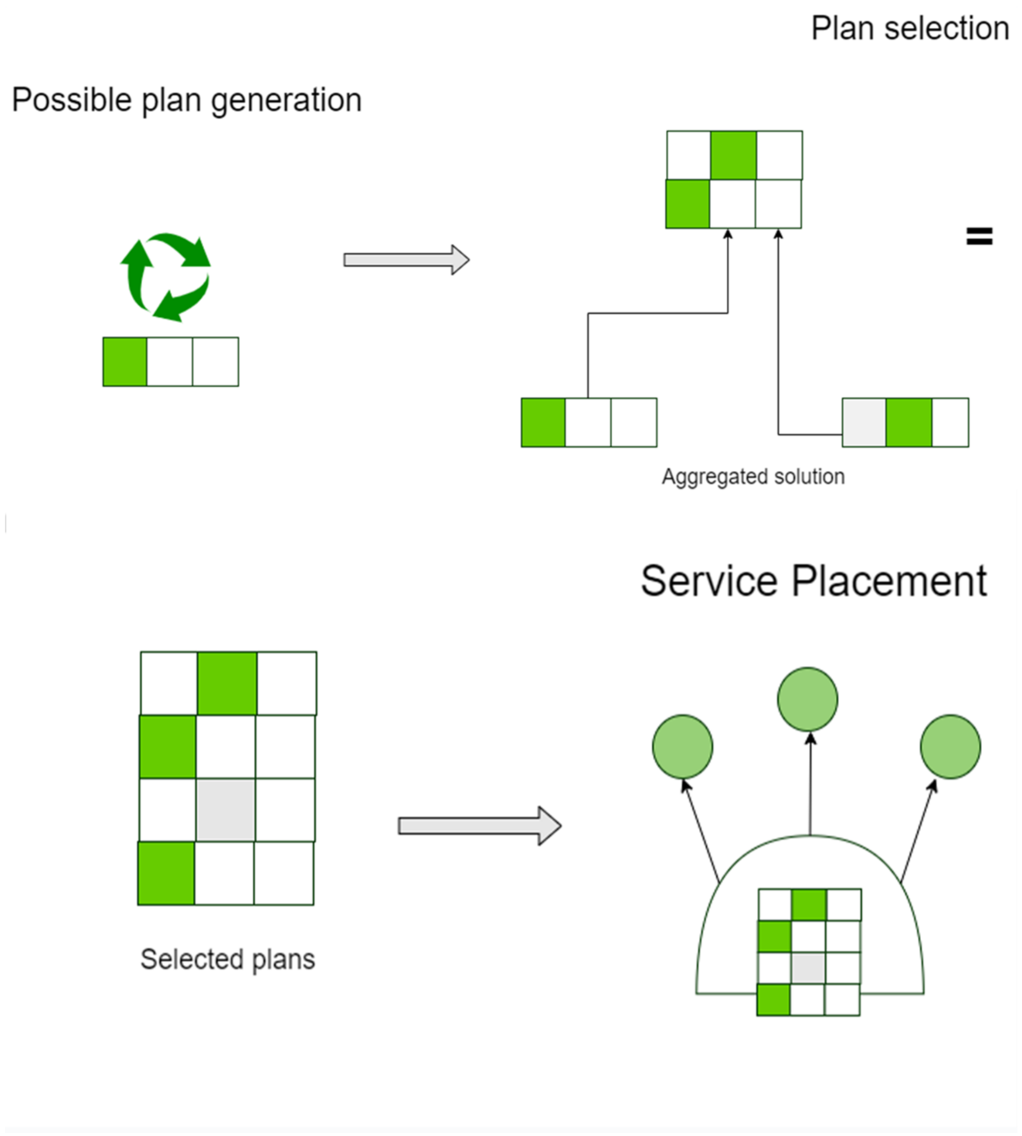


Figure 1: Plan generation and selection for service placement.

IoT service providers need to consider a wide range of factors when selecting an IoT platform, including:

1. The specific service requirements of those devices and applications.

2. The scalability and capacity requirements of the platform.
3. The security and privacy requirements of the platform.
4. The cost of the platform.
5. The types of devices and applications that will be connected.

Once an IoT platform has been selected, service providers need to configure the platform to support the specific devices and applications that will be connected. This includes provisioning devices with the necessary credentials and configuring the platform to expose the required services. Service providers also need to consider how they will manage the IoT platform over time. This includes managing updates and patches, as well as monitoring platform performance and capacity.

The edge-cloud system is a new type of cloud computing system that combines the benefits of both edge computing and cloud computing. It enables organisations to process data at the edge of their networks, close to where it is generated, and then store and analyse it in the cloud. This system provides a number of advantages over traditional cloud computing systems, including improved performance, reduced latency, and increased security. If we consider a cloud computing environment, it refers to direct access and communication which sets the nodes mapped as the nearest neighbours of the cloud. Before considering Edge/Fog environment as a possibility we take in account the inter-communication between node networks created.

1.5 Organization

The report is organised as follows:

- Chapter-02 outlines the existing related work in the field of Edge-Fog computing and IoT service placement in the edge-to-cloud infrastructure. It further presents the outputs which we eventually compare and discuss in this report.
- Chapter-03 puts forward the system that is formulated to cater the IoT service placement problem and is designed to work so as to reduce

latency. This is where we cover the software requirement and load balancing factors.

- Chapter-04 puts forward the analysis of the results in depth and also with content to existing work in the field.
- Finally, Chapter-05 presents the conclusion of the study. It also contains the application contribution with future scope.

Chapter-2
LITERATURE SURVEY

In this section the authors have covered various studies related to edge user allocation problems [8] and solutions pertaining to the same. One of the key challenges in fog computing is how to efficiently allocate resources to edge users. This is especially challenging when the number of edge users is large and dynamic. Various resource allocation schemes have been proposed, but there is no clear consensus on which is the best approach. The edge user allocation problem is a problem that arises in the context of allocating users to edges in a network. The problem is to find a mapping of users to edges such that the sum of the weights of the edges allocated to each user is minimised. The problem is NP-hard [9]. IoT devices are becoming increasingly common, with many organisations using them to monitor and manage their operations. However, due to the large number of devices and the variety of services they offer, load-balancing IoT service placement is becoming a challenge. One approach to solving this problem is to use a software-defined network (SDN) controller to dynamically adjust the placement of IoT services based on the current load. This would allow the system to automatically adjust the placement of services as the number and type of devices change, and as the load on the system changes. Another approach is to use a central database that keeps track of the number and location of devices, as well as the load on each device. This database could then be used to determine the best placement of services based on the current load. Whichever approach is used, it is important to consider the trade-offs between flexibility and performance. For example, if the system is too flexible, it may take longer to find an optimal solution. On the other hand, if the system is not flexible enough, it may not be able to adapt to changes in the environment.

To reduce the delay suffered by services while adhering to capacity restrictions, the authors in [10] introduce a QoS-aware service allocation for fog environments. A multi-dimensional knapsack problem is used to describe this goal, with the goal of simultaneously minimising the overall service execution delay and the load on the edge nodes. In [11], authors present a two-step resource management approach with the goal of using the fewest possible edge nodes while reducing the amount of time needed to deliver services. For each device, a pool of backup edge nodes and a home edge are first chosen. Finding the edge nodes that have the lowest latency between them and that device is their goal. The specified edge nodes are then used to host the necessary IoT services, ensuring the desired response time. A different project with the same objective as the ones listed in [11] and [10] has been proposed by researchers in [12]. According to a backtrack search algorithm and related heuristics that serve the goal, the recommended mechanism selects locations. The authors of [13] have offered a conceptual framework for service placement for the edge-to-cloud system. Their objective is to increase edge node use while taking user constraints into account by using a genetic algorithm for optimization. In order to take advantage of Internet of Things nodes for IoT service execution, the authors introduce the concept of a fog cell, which is software that runs on IoT nodes. An edge-to-cloud control middleware that oversees the fog cells has also been introduced. Any associated fog cells or other control nodes are under the supervision of a fog orchestration control node. The latter enables IoT services to be managed separately from cloud nodes.

In the rapidly evolving field of fog computing, resource allocation is a major challenge that researchers have been addressing. One of the main issues in fog computing is to allocate resources to edge users efficiently. The edge user allocation problem arises in the context of mapping users to edges in a network. The objective is to minimize the sum of the weights of the edges allocated to each user. This problem is NP-hard and becomes increasingly challenging as the number of edge users is large and dynamic.

To address this problem, various resource allocation schemes have been proposed. However, there is no clear consensus on which approach is the most effective. For instance, one approach involves the use of a software-defined network (SDN) controller to dynamically adjust the placement of IoT services based on the current load. The SDN controller would allow the system to automatically adjust the placement of services as the number and type of devices change, and as the load on the system changes. Another approach involves the use of a central database to keep track of the number and location of devices, as well as the load on each device. This database could then be used to determine the best placement of services based on the current load.

Regardless of the approach used, it is important to consider the trade-offs between flexibility and performance. For example, a system that is too flexible may take longer to find an optimal solution, while a system that is not flexible enough may not be able to adapt to changes in the environment. Therefore, researchers have been exploring different solutions to this problem, such as QoS-aware service allocation, multi-dimensional knapsack problem, two-step resource management, and genetic algorithms.

In computing networks, there may be instances where certain nodes experience low levels of activity while others are overwhelmed with the entire network load. This load imbalance can lead to various issues such as system and network failures, increased energy consumption, and longer execution times. To prevent such problems, load balancing is essential to distribute the load evenly across all resources based on their capacity. This ensures that no resources are underutilized or overburdened in a fog environment. Load balancing is also necessary for cloud data centres to ensure efficient workload distribution, optimal functioning, and prevention of overload and deadlock issues.
The primary objective of load balancing is to distribute a significant amount of data on servers, thus allowing resources to be utilized effectively. It possesses various characteristics, such as even workload distribution, efficient resource usage, improved system performance, reduced energy consumption, enhanced user satisfaction, and shorter response times. Load balancing functions include the efficient distribution of network loads or client requests across multiple servers, active nodes that respond to end-user requests for high availability and reliability, and server flexibility that permits the addition of servers to the network when required.

The increasing use of IoT in real-time applications has led to a greater need for load balancing in the fog environment. Load balancing can improve resource utilization and user satisfaction, leading to enhanced system performance and reduced operational costs. It ensures that resources are not overloaded or underutilized by distributing the workload evenly among processors. In cloud-fog architecture, fog nodes receive bulk requests from users, and the load balancer at the fog layer is responsible for distributing the tasks among processing nodes.

The primary goal of load balancing is to address the challenges faced by overloaded resources in the fog layer. Tasks assigned to virtual machines may be either reliant or independent on them, and the load is categorized based on CPU, storage device, and network load. Load balancing involves detecting overloaded and underutilized nodes and redistributing the workload among them. Proper utilization of fog resources can improve system performance, and these resources can be either hardware or virtual. The load balancer allocates tasks to physical machines, which in turn assign them to virtual machines, and transfers the workload between physical or virtual machines.

In one study, researchers introduced a QoS-aware service allocation for fog environments that aimed to minimize overall service execution delay and the load on the edge nodes. They used a multi-dimensional knapsack problem to describe this goal. In another study, a two-step resource management approach was proposed with the goal of using the fewest possible edge nodes while reducing the amount of time needed to deliver services. A pool of backup edge nodes and a home edge were first chosen for each device, and then the edge nodes with the lowest latency between them and that device were selected. These nodes were used to host the necessary IoT services to ensure the desired response time. Another project with similar objectives proposed a backtrack search algorithm and related heuristics to select locations.

Researchers also proposed a conceptual framework for service placement in the edge-to-cloud system, with the aim of increasing edge node usage while taking user constraints into account. They used a genetic algorithm for optimization, which helped to manage IoT services separately from cloud nodes. Additionally, the concept of a fog cell, which is software that runs on IoT nodes, was introduced to take advantage of IoT nodes for IoT service execution. An edge-to-cloud control middleware that oversees the fog cells was also proposed, along with a fog orchestration control node that supervised any associated fog cells or other control nodes.

Author in [14] main goal is to increase the number of edge node-served services while maintaining QoS standards such as response time. They employ an algorithm to overcome the issue that makes use of validation, rounding, and relaxation. Authors in [15] offer a service placement strategy that maximises the amount of services assigned to edge nodes, similar to the earlier efforts [16], [17]. The suggested method uses context data from the edge nodes, such as location, response time, and resource consumption, to distribute services. Workload distribution is defined by [18] as an interaction between edge-to-cloud nodes. Investigated and roughly resolved is the trade-off between power usage and transmission delay in the interaction. A relevant framework for understanding the cooperation between edge-to-cloud nodes is provided by simulation and numerical results. For a three-layer fog-cloud architecture made up of the fog device, fog server, and cloud layers, authors in [19] present for resource allocation. The processing time, bandwidth, and reaction time of the available resources are ranked according to three factors in order to address the time constraints imposed by dynamic user behaviour in resource provisioning. These resources are then distributed in a hierarchical and hybrid way according to the requests that were received. DRAM is a different load-balancing resource allocation mechanism that authors have [20] offered. DRAM uses service migration after allocating network resources statically to create a dynamically balanced workload across edge nodes.

In order to supply IoT services, authors in [21] create an Integer Linear Programming (ILP) problem that balances two goals: minimising deployment cost (which includes the costs of computation, memory, and data transfer) and raising service acceptance rate. Greedy Randomised Adaptive Search techniques [22], which iteratively minimise the provisioning cost while load-balancing networked nodes, are used in the suggested solution. In order to reduce the processing time of compute tasks in fiber-wireless enhanced vehicle edge computing networks, authors in [23] suggest a task offloading architecture. Two strategies based on software-defined networking and game theory are given to achieve the load-balancing of the computation resources at the edge servers. For each vehicle to successfully complete its computation task, these schemes, namely a nearest offloading algorithm and a predictive offloading algorithm, optimise the offloading decisions for local execution, offloading to a Multi-access Edge Computing (MEC) server connected to roadside units, and offloading to a remote cloud server. In the table 1, the authors have made a drawn a comparison to distinguish existing work in different categories including QoS, Load Balancing, Techniques employed and whether the Distributed Network as used or not.

The emergence of the Internet of Things (IoT) and the ubiquitous adoption of smart devices have transformed virtually every industry, making it imperative to provide advanced services that are scalable, reliable, and high-performing. The integration of IoT and Cloud Computing (CC) has given rise to cloud IoT, a new paradigm that aggregates, stores, and processes IoT-generated data. While cloud IoT brings immense opportunities, it is also constrained by bandwidth, latency, and connectivity issues. This has led to the development of Edge and Fog Computing (FC), where computing and storage resources are located at the edges, closer to the source of data. The hierarchical and collaborative edge-fog-cloud architecture brings significant benefits, as it enables the distribution of computation and intelligence, including AI, ML, and big data analytics, to achieve optimal solutions while satisfying constraints such as the delay-energy trade-off.

Despite the advantages of edge-fog-cloud computing, its implementation poses several challenges, including design, deployment, and evaluation. To provide a comprehensive understanding of this paradigm, this paper presents an in-depth tutorial and discusses the main requirements, state-of-the-art reference architectures, building blocks, components, protocols, applications, and other similar computing paradigms. The paper also presents a holistic reference architecture for edge-fog-cloud IoT, discussing the major corresponding design and deployment considerations, including service models, infrastructure design, provisioning, resource allocation, offloading, service migration, performance evaluation, and security concerns.

In addition to these considerations, the paper also explores the role of privacy-preserving, distributed, and collaborative analytics, as well as the interaction between edge, fog, and cloud computing. Finally, the paper reviews the main challenges in the field of edge-fog-cloud computing that need to be tackled to realize the full potential of IoT.

Several studies have investigated the integration of IoT and CC, resulting in the development of cloud IoT. However, cloud IoT faces challenges such as latency, connectivity, and bandwidth. To overcome these challenges, edge and fog computing have emerged, offering distributed computing and storage resources closer to the data source. This hierarchical architecture enables the distribution of computation and intelligence, leading to optimal solutions while satisfying constraints such as the delay-energy trade-off.

Despite the benefits of edge-fog-cloud computing, several challenges remain, including design, deployment, and evaluation. This paper provides a comprehensive insight into the paradigm by presenting a tutorial and discussing various aspects, including reference architectures, building blocks, components, and protocols. Additionally, it explores the role of privacy-preserving, distributed, and collaborative analytics, as well as the interaction between edge, fog, and cloud computing. Finally, the paper reviews the main challenges in the field of edge-fog-cloud computing that need to be addressed to fully realize the potential of IoT.

In conclusion, this paper [24] presents a thorough literature review of edge-fog-cloud computing, highlighting the benefits and challenges of this paradigm. It offers a comprehensive understanding of the underlying technologies and presents a holistic reference architecture for edge-fog-cloud IoT. By discussing the major design and deployment considerations and exploring the role of privacy-preserving, distributed, and collaborative analytics, this paper provides opportunities for more holistic studies and accelerates knowledge acquisition in the field.

The paper identifies the dynamic service placement problem, which addresses the adaptive configuration of application services at edge servers to facilitate end-users and those devices that need to offload computation tasks. The paper presents a systematic literature review of existing dynamic service placement methods for MEC environments from networking, middleware, applications, and evaluation perspectives. The review reveals research gaps in the big picture and identifies eight research directions that researchers follow.

With the advent of cloud-based applications such as mixed reality, online gaming, and healthcare, there is a need for efficient infrastructure management to provide a cloud-like environment for end-users. MEC extends the cloud computing paradigm and leverages servers near end-users at the network edge to provide a cloud-like environment, but the optimum placement of services on edge servers plays a crucial role in the performance of such service-based applications.

The review [25] then investigates dynamic service placement methods from a middleware viewpoint, which includes different service packaging technologies and their trade-offs. The review categorizes the research objectives into six main classes, proposing a taxonomy of design objectives for the dynamic service placement problem. The paper also introduces the applications that can take advantage of dynamic service placement and investigates the evaluation environments used to validate the solutions, including simulators and testbeds. Finally, the paper compiles a list of open issues and challenges categorized by various viewpoints. Overall, this literature review provides a comprehensive insight into the dynamic service placement problem in MEC environments and identifies future research directions.

QoS is the primary concern in dynamic service placement methods from an application viewpoint, including QoS levels and factors. Application QoS can typically be categorised into three levels . The first level is guaranteed services (hard QoS) that have strict hard real-time QoS guarantees. This level is suitable for safety-critical applications such as remote surgery. The second level is soft QoS that does not require hard real-time guarantees but needs to reconfigure and replace failed services. Finally, the last level is the best effort, where there are no guarantees when a service fails. According to the surveyed papers, time-related QoS factors receive much attention compared to others. Some of these factors, such as application response time and user-perceived latency, are applied to both soft QoS and hard QoS. Other factors such as the worst application completion time and the number of applications in outage focus on hard QoS. The next group of QoS factors concentrates on throughput and resource utilisation, namely processing, network, and energy resources. It shows how effectively the edge nodes are being used and that the load is being spread evenly across them, and no one edge node is overloaded. Modern applications, such as augmented reality and autonomous vehicles, have massive network throughput. Energy efficiency is a concern to both users and edge infrastructure providers. Security is another QoS factor that is addressed in a few work. With the new legislation, such as GDPR, privacy concerns are becoming as essential as other security factors when developing a service placement method. MEC enables the processing of exabytes of data near where it is required and generated. Such proximity benefits applications from different domains as they can address challenges regarding data volume, interoperability, and latency. In the following, we review these application domains that can benefit from dynamic service placement mechanisms to address these challenges effectively and efficiently use the available resources in MEC architectures.

The agent-based approach is a real-time strategy that involves assigning tasks to servers based on their current load, as determined by their respective agents. [45] proposed an agent-based automated service composition (A2SC) technique for resource provisioning in cloud computing, with a focus on reducing virtual machine costs and ensuring equal resource distribution across four data centres with different platforms. They employed Java to obtain experimental results and aimed to provide efficient service allocation in the data centres.

[26] proposed a multi-agent-based offloading technique for mobile fog computing, which uses reinforcement learning to minimize service delivery latency to mobile users. The mobile codes are deployed on geographically distributed mobile fogs, with agents serving as entities that have prior knowledge of the environment and learn from it. The goal is to reduce execution time and improve mobile user access to services, with simulation results obtained using OmNet++.

[27] developed an agent-based task assignment approach for load balancing in cloud computing, incorporating principles of fair competition and dynamic adjustment for task allocation to improve resource allocation and utilization. They used CloudSim to obtain simulation results, which showed an increase in processing time with this technique.

In 2017, [28] introduced cooperative load balancing (CooLoad) for fog computing environments. The approach involved forwarding requests to another node when a processing node was already fully occupied, resulting in improved quality of service and equal load distribution.

Similarly, [29] proposed a scalable and dynamic load balancer (SLDB) for mobile edge computing. The SLDB algorithm employed minimal perfect hashing to optimize performance and reduce memory consumption. The Data

Plane (DP) of SLDB considered fitting data into the cache to improve processing speed.

In 2018, Rafque et al. [30] introduced a new bio-inspired hybrid algorithm (NBIHA) for load balancing in fog environment. This algorithm reduces the average response time and energy consumption by employing efficient task scheduling. The proposed system architecture has three layers: the client layer, the fog layer containing the scheduler, and the cloud layer with data centers. The scheduling of tasks is performed using meta-heuristic particle swarm optimization (MPSO), and simulation results are obtained using iFogSim. Arshad et al. [31] evaluated and analysed two nature-based algorithms called Pigeon-inspired optimization (PIO) and Binary bat algorithm (BBA) for reducing the energy consumption of cloudlets. A three-layer architecture has been proposed, with smart homes in the first layer, cloudlets in the second layer, and cloud servers in the third layer. The energy consumption of smart meters at smart homes is measured for bill estimation.

In 2019, Fahs et al. [32] proposed a routing algorithm for the fog environment to reduce latency and ensure equal load distribution. The proposed proximity-aware system is based on Kubernetes, and it helps to reduce the service access time from sender to receiver by equally distributing the load. Javaid et al. [33] proposed a cuckoo search load balancing algorithm, which uses a combination of Levy walk distribution and flower pollination to optimize the response time and processing time of fog and cloud environments. Cost is also considered, and efforts are made to reduce the cost of data transfer, microgrids, VMs, and the total cost. Khattak et al. [34] proposed a fog-cloud server-based architecture to achieve proper utilization of all resources in E-healthcare. They aimed to distribute equal load among all servers by shifting the load from overloaded servers to the ones having less load. Parameters like latency, load balancing, QoS, and bandwidth were considered, and simulation results were obtained using iFogSim.

In 2020, Talaat et al. [35] proposed a resource allocation-based load balancing approach that uses reinforcement learning to handle incoming requests by measuring server loads. The workload is distributed among all available resources for their appropriate utilization, and a three-layer fog-cloud-based architecture is proposed for health care. Kaur et al. [36] proposed a load balancing approach based on the equal distribution of workload in a three-tier architecture of fog-cloud to reduce energy consumption, cost, and processing time in the fog-cloud environment. The proposed approaches are implemented and compared with existing round-robin and throttled algorithms using a cloud analyst simulation tool. Bhatia et al. [37] proposed a quantumized approach to task scheduling in the fog environment, which distributes the workload among all fog nodes to improve system performance and reduce execution delay. iFogSim is used to show simulation results.

In [1,2, 3, 4, 5, 12, 13,43,44], the authors note that Both QoS and load balancing are not considered except in one case of [44], Meanwhile the technique and simulations are very diverse.

Table 1: Comparison of relevant literature.

Author(s)	**Quality of Service**	**Load Balancing**	**Methodology**	**Distributed**
Mahmud et.al [1]	✓	-	Deadline based dynamic allocation	-
Li et.al [2]	✓	✓	Clustering & Hierarchical Load Balancing	-
Ferretti et.al [3]	✓	-	Workload Allocation Based on delay & power consumption	-

Aburukba et.al [4]	✓	-	Task Placement Based on latency, energy consumption & operational cost	-
Songhorabadi et.al [5]	✓	✓	Software Defined Networking(SDN) based on load balancing task offloading	-
Yousefpour et.al[43]	✓	-	FOGPLAN	-
Xia et.al [12]	✓	-	First Fit	-
Skarlat et.al [13]	✓	-	Decentralized approach to data processing and resource provisioning	-
Kapsalis et.al [44]	✓	✓	Fog Based Implementation	✓

The review of literature on dynamic service placement methods shows that only a few of these methods adopt specific service packaging techniques. Moreover, only a small fraction of these methods consider the costs associated with transferring service instances, such as the time required to download instances, the necessary bandwidth, and launch time. This lack of attention to these critical factors raises concerns about the practicality of these methods. The survey also indicates that most methods assume heterogeneity in both edge servers and services, but they do not give enough attention to privacy and security issues and service inter-operation. These research directions are crucial, especially in light of the increasing use of micro-service architecture and growing concerns over GDPR. The analysis of design objectives indicates that the most popular objective is improving service-level QoS, with most researchers proposing additional objectives alongside QoS improvement. However, it is also essential to investigate the method's overheads on various resources, such as processing, bandwidth, and energy and minimize them as secondary objectives. From a resource management perspective, the processing resource is the most critical, while energy and memory are the least focused. With the growing trend towards reducing the carbon footprint by cloud infrastructure and edge servers, it is vital to investigate the energy efficiency of dynamic service placement methods.

Chapter-3
SYSTEM DESIGN & DEVELOPMENT

In this section, application model is introduced for improving load balancing and Quality of Service (QoS) in the Fog-Cloud environment. A simple working of fog computing technique is displayed in Figure 2. Furthermore, the overall system architecture for resolving the problem is also described.

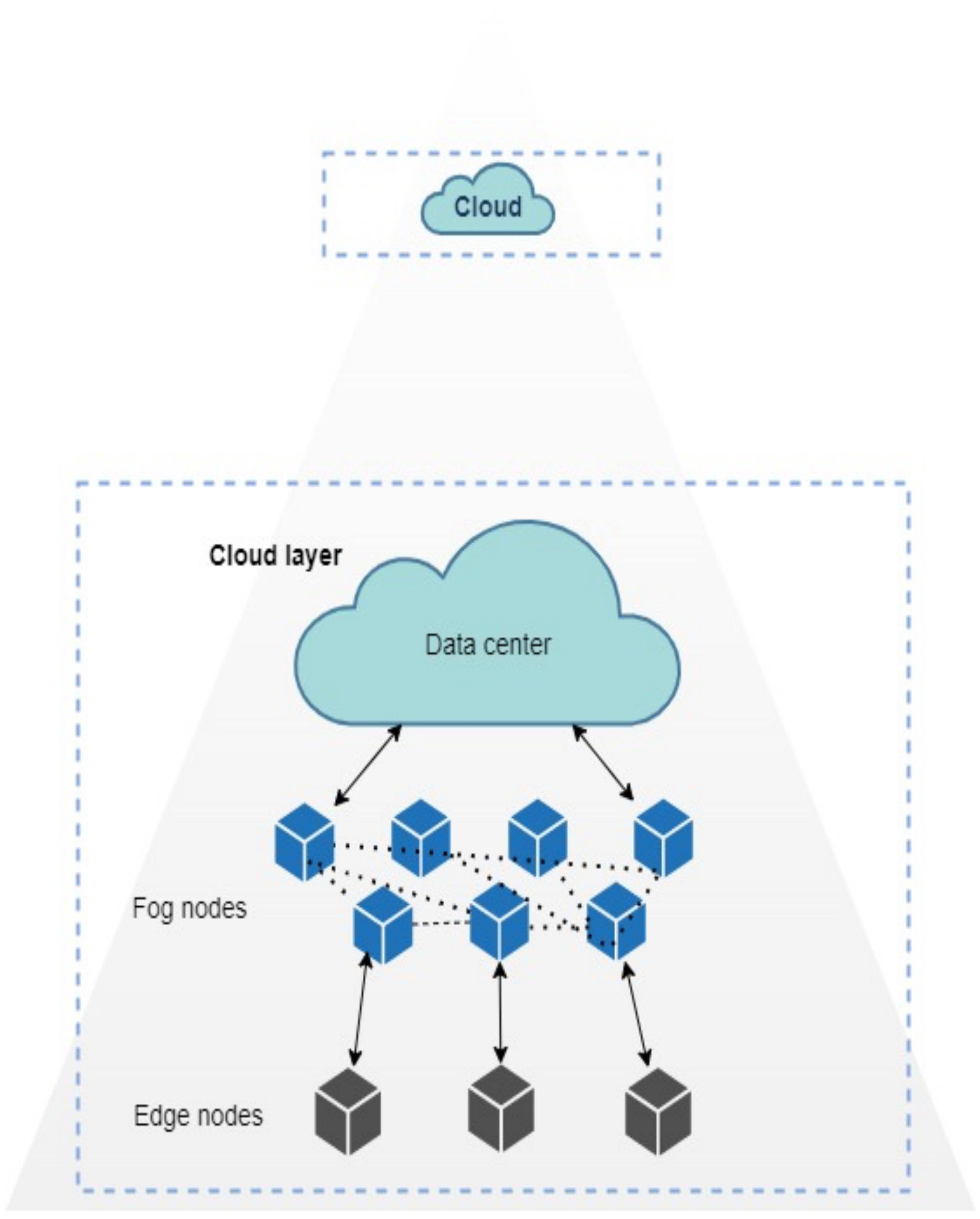


Figure 2: Basic working of edge computing

3.1 Application Scenario

In this section step wise implementation and comparative study procedure is explained:

a) Create infrastructure using multiple networks with heterogeneous edge-to-cloud nodes.
b) Receive input workload (IoT services/tasks).
c) Distribute input workload over edge nodes.
d) Implement the proposed technique by generating potential plans for each node.
e) Use the First Fit strategy.
f) Use the Cloud approach.
g) Keep track of outcomes for analysis (for the evaluation several varied parameters are considered that consist of: size of the network, type of topology, workload distribution method, lambda values, and host proximity).

3.2 Dataset

The simulation considers Google Borg compute clusters trace v3 and divides it into a 5-min time interval [27]. The dataset presents:

- Informative histograms on CPU usage (per 5-minute period)
- Data about alloc sets (job specific shared resource reservations)
- Explanation on master/worker relationship

An alloc set is a set of reserved resources (alloc instances) into which jobs can be scheduled, by placing a job's tasks inside the alloc instances. Alloc sets represent 20% of the total CPU allocations and 18% of the RAM. 15% of the jobs are marked to run in an alloc set, most of which (95%) are from the production tier. Jobs within allocs have a higher average memory utilization (73%) than other jobs (41%) [42].

Table 2: Symbols Used in the Study.

Symbol	Meaning
Δ	Binary decision w.r.t. fog nodes
F	Set of fog nodes available in the given network
f	Fog node
Θ	Total available resources
$\Phi_{t,\rho}$	Total CPU utilization for plan ρ
$\Phi_{m,\rho}$	Total memory utilization for plan ρ
M	Overall memory utilization (in bytes)
T	Overall CPU utilization (in MIPS)
m	Memory utilization per plan
t	CPU utilization per plan
z	Traffic rate (in MIPS)
C	Processing capacity of fog nodes
R	Total services available
r	Set of services requested from IoT devices
ρ	Best possible solutions
P	Set of service allocation plans
j	Iterations to get best possible local solution
p	Solution in iteration j
κ	Latency vector for placement algorithm
κ_i	Latency per placement plan
ω	Set of waiting time for r services
ω_i	Waiting associated to ith requested service
W	Waiting time for dynamic set of services
$\Phi_{c,\rho}$	Total cost for plan ρ

The 5-minute recording intervals are taken as periods and divided into small sets for simulation purposes. Here it has been assumed and configured deadlines in accordance with the assumed IoT devices in the given dataset. Number of various services considered are 40, numbered (0-39) and the target edge node to which the task is propagated. The tasks are going to be input to the networks with different settings. The symbols used in the study are shown in Table 2. We have 3 network graphs with the size set {200,400,1000}. So, the tasks are assigned to different edge node's id based on the given network size. Borg measures CPU in internal units called "Google compute units" (GCUs): CPU resource requests are given in GCUs, and CPU consumption is measured in GCU-seconds/second. One GCU represents one CPU-core's worth of compute on a nominal base machine. Table 3 presents a overall view of the dataset.

The squared coefficient of variation (2) [43] where:

$$\sigma^2 = \text{variance}/\text{mean}^2$$

The high σ^2 means that the dataset positions probability of high queueing delay even when the system load is low. It is known that when σ^2 is high, there is a wide range of job sizes, which helps us test our proposed solution in a variable environment.

The real production traces assist in informing job designs on the basis of common jobs and resource requirements. On clustering similar jobs to study their nature, the dataset reflects on its ability to cater while demonstrating scheduling techniques which handle large spikes in job arrival time. The dataset considers CPU and memory utilization as they are bottleneck resources [27].
Google cluster trace is used to identify the resource capacity for each node. With 12,500 workstations, the Google cluster can handle 12 MB jobs. The 89354 tasks in the profiles that were taken into account for this evaluation. Each network contains 11 different sorts of nodes: "10 varieties of fog nodes + one type of cloud node. 'One edge node is assigned to each input job from the

dataset. The task's assigned node's id is already set using beta distributions and random numbers.

Machine types, their probabilities, and capacities regarding to Google cluster data:

Table 3: Google Cluster database categorization for node network.

Class	Type	Colour	CPU	Memory	Probability
Class B1	Fog	Colour: pink	CPU: 0.5	MEM:0.03	prob= 0.0004
Class B2	Fog	Colour: pink	CPU:0.5	MEM:0.06	prob=0.00008
Class B4	Fog	Colour: violet	CPU:0.5	MEM:0.12	prob=0.00416
Class A	Fog	Colour: violet	CPU:0.25	MEM:0.25	prob=0.01008
Class B6	Fog	Colour: purple	CPU:0.5	MEM:0.25	prob=0.30904
Class B7	Fog	Colour: purple	CPU:0.5	MEM:0.5	prob=0.53456
Class B5	Fog	Colour: blue	CPU:0.5	MEM:0.75	prob=0.08008
Class B3	Fog	Colour: blue	CPU:0.5	MEM:0.97	prob=0.0040
Class C2	Fog	Colour: blue	CPU:1	MEM:0.5	prob=0.00024
Class C1	Fog	Colour: blue	CPU:1	MEM:1	prob=0.0636

3.3 System Description

The authors have formulated the problem for a given set of targeted IoT services which have certain defined requirements which correspond to the edge-to-cloud nodes, both of which will be mapped according to the best suited environment while minimizing the cost-of-service execution. The authors assume minimal information about the IOT devices considered in this network. The Resource accessing mechanism for IoT devices in Fog- Edge environment is shown in Figure 3.

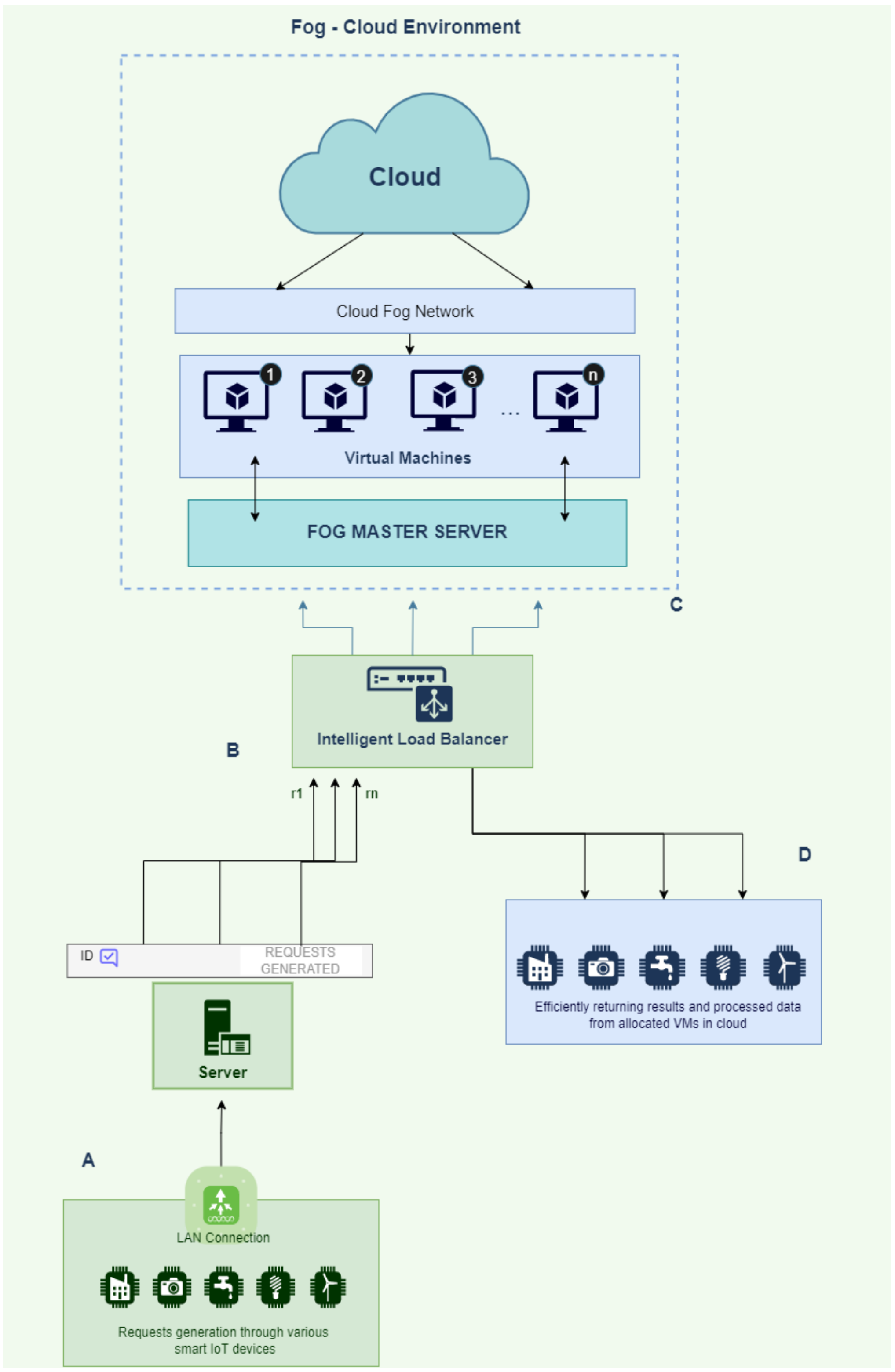

Fog - Cloud Environment
Cloud
Cloud Fog Network
1
2
3
n
Virtual Machines
FOG MASTER SERVER
C
Intelligent Load Balancer
B
r1
rn
D
ID
REQUESTS GENERATED
Server
Efficiently returning results and processed data from allocated VMs in cloud
A
LAN Connection
Requests generation through various smart IoT devices

Figure 3: Resource accessing mechanism for IoT devices in Fog- Edge environment.

Avoidance of overloaded or under loaded nodes result in better outcomes and balanced network while improving QoS.
The request handling flow works in the following manner:

1. Establishing connection between base server and IoT devices via LAN
2. Collected requests being redirected towards load balancer for distribution of incoming workload
3. Edge and Fog architecture as a part of cloud computing environment, to which requests are & being redirected to fulfil the purpose as shown in Figure 2.
4. Returning the computed results back to respected server connected to IoT devices

We deal with overall cost which consists of three major factors which are:

(i) number of deadline violations,

(ii) number of services unavailable, and (

(iii) traffic management.

The first two effect the Quality of Service (QoS) while the last one implicates poor network performance. So, we take:

$$\Phi c, \rho: totalcostforplan\rho$$

Where plan ρ refers to an outcome.
Such stringent latency requirements are high in demand with growing network technology and user density in populated areas. The mapping of IoT and mobile devices' requests with different resource demands from the various heterogeneous servers around can be an extensive task considering that scalability is another factor in case. IoT devices not only generate latency-sensitive traffic but also have variations. Hence such services need to be held and run near data sources which is achieved by the fog-edge cloud environment. These fog nodes have certain defined capacity to host requested services and run to expand the edge network by cooperating and work with independent IoT services (as assumed here).

The demands are sent from fog nodes to cloud servers over a WAN that spans a sizable geographic area from the edge to the core network. Contrarily, IoT queries are sent across a local area network to edge nodes, which are typically situated close to end devices (LAN). In contrast to the WAN, the LAN communication delay might therefore be eliminated.

The total cost function is defined around two quantities which are: memory utilization and CPU utilization. This can be defined as:

$$\Phi_{c,\rho} = \Phi_{t,\rho} + \Phi_{m,\rho}$$

Assuming suitable deadline range for IoT service in a given sector,

$$K_{\mathrm{i}}: latencyperplacementplan$$

$$K: latencyvectorforplacementalgorithm$$

To bring forward a solution to cover these grounds with the goal of improving overall placement optimization and service execution, we consider following constraints for hosting on respected r

1. **Limiting constraint** to insist on at most one allocation per service

$$0 \leq \sum_{i=1}^{|F|} \Delta_i \leq |\mathrm{R}||r|$$

2. **Placement constraint** to ensure total placement decisions must be less than available resources

$$\sum_{i=1}^{|r|} \Phi_{\mathrm{c,\rho}} (\Delta_{\mathrm{i}}) < \Theta \quad \forall \mathrm{fj} \in \mathrm{F}$$

We consider these two metrics for overall placement consideration with desired QoS parameters being handled. Furthermore, we understand the waiting time (w_i) accounts the time period between receiving a placement request and decision making for its allocation. r_i service requests that are received are forwarded to the fog node f_j for expected decision making on placement. This involves queuing of these requests. So, we define another constraint for stable queuing and handling in load balancer for fog node f_j :

$$z_{f,i} < C_{f,i}$$

where $z_{f,i}$ is traffic rate for the given fog node and $C_{f,i}$ represents processing capacity of that particular fog node.

Final Optimization:

$$\min(\Phi_{c,\rho} + \Delta) = \min(\Phi_{t,\rho} + \Phi_{m,\rho})$$

3.4 Methodology

Service placement is a critical function for managing the Internet of Things (IoT). By understanding the service requirements of devices and applications, and mapping these to the capabilities of available IoT platforms, service providers can ensure that devices and applications are able to interoperate and function as intended. IoT service providers need to consider a wide range of factors when selecting an IoT platform, including:

- The specific service requirements of those devices and applications
- The scalability and capacity requirements of the platform
- The security and privacy requirements of the platform
- The cost of the platform
- The types of devices and applications that will be connected

Once an IoT platform has been selected, service providers need to configure the platform to support the specific devices and applications that will be connected. This includes provisioning devices with the necessary credentials and configuring the platform to expose the required services. Service providers also need to consider how they will manage the IoT platform over time. This includes managing updates and patches, as well as monitoring platform performance and capacity.

The rise of the Internet of Things (IoT) and the explosion of data generated from connected devices have created a need for a more efficient computing system that can handle the massive amounts of data generated. The traditional cloud computing model, where data is sent to a centralized location for processing and analysis, has limitations in terms of latency and bandwidth. As a result, a new computing paradigm called the edge-cloud system has emerged, which combines the benefits of both edge computing and cloud computing.

In an edge-cloud system, data is processed and analysed at the edge of the network, closer to where it is generated. This is done by deploying small, low-power computing devices, called edge nodes, at the network edge. These edge nodes are responsible for processing and analysing the data in real-time, and then sending it to the cloud for storage and further analysis.

One of the major benefits of the edge-cloud system is improved performance. By processing data at the edge, organizations can reduce the amount of data that needs to be sent to the cloud, thereby reducing the latency and bandwidth requirements. This results in faster response times and improved overall system performance.

Another advantage of the edge-cloud system is increased security. By processing and storing data at the edge, organizations can reduce the risk of data breaches and other security incidents. This is because the data is stored and processed locally, rather than being sent to a centralized cloud, which is a more attractive target for hackers.

Furthermore, the edge-cloud system is also beneficial for reducing energy consumption. By processing data at the edge, organizations can reduce the amount of data that needs to be sent to the cloud, which reduces the energy consumption required for transmitting the data.

In conclusion, the edge-cloud system is a new computing paradigm that combines the benefits of both edge computing and cloud computing. It enables organizations to process and analyse data at the edge of their networks, closer to where it is generated, and then store and analyse it in the cloud. This system provides a number of advantages over traditional cloud computing systems, including improved performance, reduced latency, increased security, and reduced energy consumption. As the amount of data generated from connected devices continues to grow, the edge-cloud system will become increasingly important for organizations looking to improve their computing infrastructure.

As shown in Figure 4, cloud-edge architecture, we receive input requests from various placed IOT devices. These requests sent by the client server are primarily processed by the load balancer at the periphery of the cloud environment. These requests are taken as input parameters in the process of creating an optimized plan in a node network. Multiple recursive calls generate an array of placement solutions to cater the QoS parameters considered which are memory utilization and cost efficiency. The intelligent load balancer creates various plans and selects the most efficient work load balance solution. The selected plan is then implemented while processing and returning the results of these requests back from the cloud.

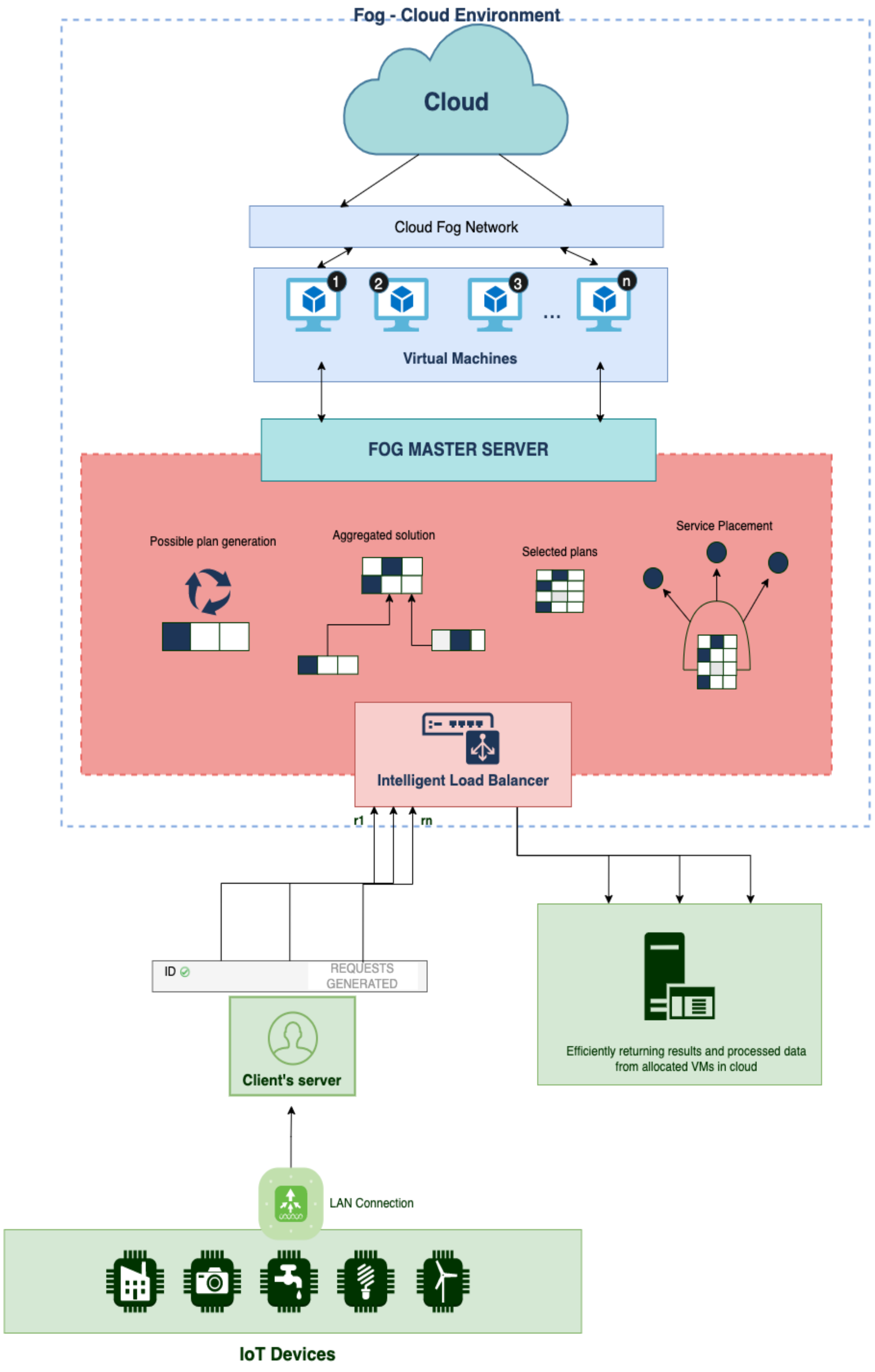


Figure 4: Proposed architecture & algorithm placement in a Fog-Edge environment.

3.5 Proposed Algorithm

In this section the authors have explained the algorithm in tabular and step wise form.

Explanation for Step wise Implementation of the algorithm

1. Define a function called $service_allocation$ that takes in two inputs: a set of services requested from IoT devices, r, and a set of network nodes, n.
2. Initialize the service allocation plan, P, and the total cost, $\Phi_{c,\rho}$, to 0.
3. Initialize two variables, i and j, to 0.
4. Sort the services in r in terms of their waiting time, ω_i, in ascending order.
5. Select neighbouring nodes from n and sort them in terms of their proximity value.
6. Compute the network latency vector N by finding the minimum latency value L for each node.
7. Start a while loop that runs as long as r is not empty.
8. Select a node n[i] and a fog node fog node[j].
9. Check if fog node[j] satisfies the constraints in equations 2, 3, and 4.
10. If the constraints are satisfied, update the service placement plan P, and update the total memory utilization, M, to be the total memory utilization for n[i].
11. Update the capacity for fog node[j] by subtracting the memory utilization for the selected service.
12. If the leftover capacity of fog node[j] is less than or equal to the processing capacity, C f, j, set the binary decision variable, $\Delta[i]$, to 1 and update the service placement plan ρ.

13. Move on to the next node, n[i+1], and fog node, fog node[j+1].

14. Repeat steps 9-13 until all services in r have been allocated.

15. Return the set of best possible solutions, ρ.

Pseudocode of Proposed Algorithm

```
Input: {incoming set of requested services from IoT devices}
Output: {P: set of possible solutions}
------------------------------------------------------------
Initialize service placement plan q
h ← select |a| neighboring nodes from n;
Sort h in terms of proximity value
Initialize i, j
while(a) do
Select a and fj
if(fj satisfies constraints) then
      Update service placement plan q
      Resource Utilisation vector R[i] ← Memory Demand M
      Possible plans vector X[j] ← p (solution)
end if
else if (the cloud node (ck) has enough capacity) then
X[i]← 1
Update service placement plan q
end if
```

```
Remove ai from a and fj from h
 i++, j++
```

The algorithm presented above is a service placement algorithm designed for an edge-cloud system. The input to the algorithm is a set of requested services from IoT devices, and the output is a set of possible solutions represented by a service placement plan.

The algorithm begins by initializing a service placement plan q. It then selects a neighbouring set of nodes, denoted by h, where a denotes the incoming set of requested services from IoT devices, and n is the set of available nodes. The nodes in h are then sorted in order of proximity value, indicating their distance from the IoT devices generating the data.

The algorithm then initializes two counters, i and j, which are used to keep track of the resource utilization vector and the possible plans vector, respectively. The algorithm then enters a loop, where it selects a service from the incoming set of requested services, denoted by a, and a node from the sorted neighbouring set of nodes, denoted by fj.

If the selected node satisfies the constraints of the requested service, the service placement plan q is updated, and the memory demand of the service is added to the resource utilization vector R[i]. The algorithm also generates a possible solution, denoted by p, and adds it to the possible plans vector X[j].

If the selected node does not satisfy the constraints of the requested service, the algorithm checks if the cloud node (ck) has enough capacity to process the service. If so, the possible plans vector X[i] is updated, and the service placement plan q is updated accordingly.

Finally, the algorithm removes the selected service from the incoming set of requested services and the selected node from the neighboring set of nodes. The counters i and j are incremented, and the loop continues until all requested services have been placed in the service placement plan.

Overall, this algorithm is designed to optimize the placement of services in an edge-cloud system by considering the proximity of available nodes to the IoT devices generating the data, as well as the resource utilization of each node. By carefully selecting the appropriate node to process each service, the algorithm can improve the overall performance and efficiency of the edge-cloud system.

Chapter-4
EXPERMENTS & RESULT ANALYSIS

In computer networking, topology refers to the way in which devices are connected to form a network. Different types of network topologies can be used depending on the requirements of the network and the types of devices being used. Each type of network topology has its own advantages and disadvantages, and the choice of topology depends on the specific requirements of the network.

4.1 Topologies Considered

- **Barabasi Albert** [46]- In network science, Barabási–Albert (BA) models are a class of random graphs that have found significant use in studying the topological properties of real-world networks. These models are named for Albert-László Barabási and Réka Albert, who introduced them in two papers published in 1999 and 2000. The BA model is an example of a scale-free network, a network whose degree distribution follows a power law. The model begins with a small number of nodes (typically m0 = m = 2) that are connected to each other by edges. At each time step, a new node is added to the network. This new node is then connected to m existing nodes in the network by edges. The nodes that the new node is connected to are chosen with a probability that is proportional to their degree. The result of this process is a scale-free network with a power-law degree distribution. The exponent of the power law is determined by the parameter m. For m = 1, the exponent is 2, and for m > 1, the exponent is 3. The BA model is a generalisation of the Erdős–Rényi random graph model, which also produces scale-free networks. The BA model is more realistic than the Erdős–Rényi model, as it produces networks with a power-law degree distribution that is often seen in real-world networks.

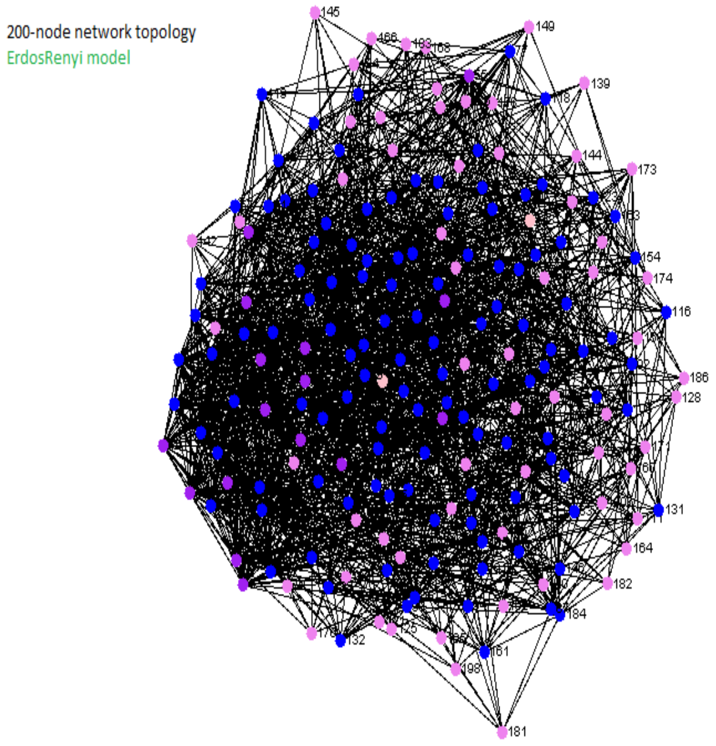


Figure 5: Erdos Renyi Model showing 200 node network topology

- **Watts-Strogatz** [37]- The Watts–Strogatz model is defined on a ring of n vertices, where each vertex is connected to its k nearest neighbours (k-1 if k is even). Each edge is then rewired with probability p, independent of the other edges. When p = 0, the resulting graph is a regular ring lattice. As p increases, the graph becomes less and less regular, until when p = 1 the graph is entirely random. The Watts–Strogatz model can be used to generate random graphs with arbitrary degree distributions, by rewiring each edge with probability p(1-k_i/2m), where k_i is the degree of vertex i and m is the sum of the degrees of all vertices.

The network graphs for the Erdos Renyi model chosen models for a 200-node network are shown in Figure 5.

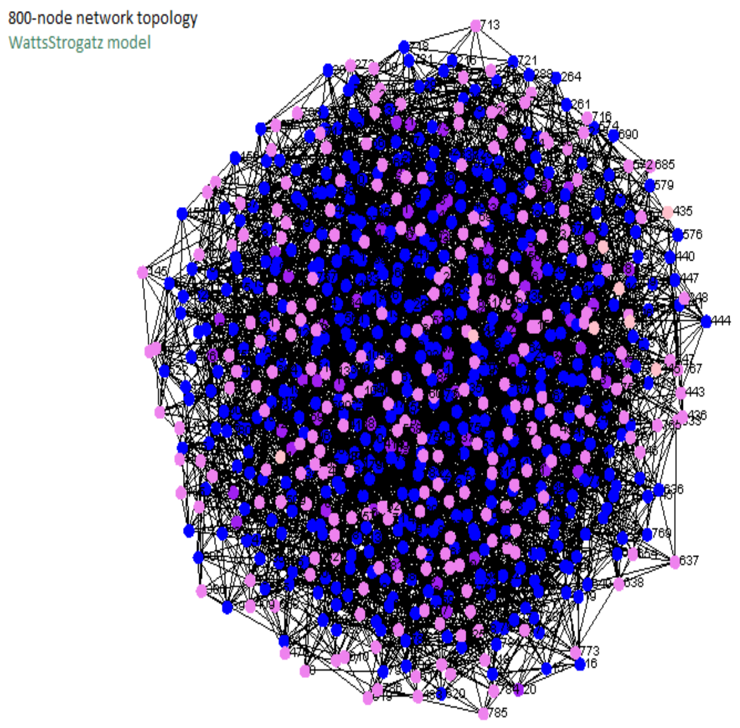


Figure 6: Watts Strogatz Model showing 800 node network topology.

- **Erdos-Renyi** [39]- In the Erdős–Rényi model of a random graph, each edge is included in the graph with probability p, independently of all other edges. The resulting graph is then a random graph on n vertices.

The network graphs for the Watts Strogatz model chosen models for a 800-node network are shown in Figure 6. Utilizing Java software that simulates a network of edge-to-cloud nodes, experimental assessment is carried out. In addition, GraphStream5 [26], a Java package, is used for graph modelling and analysis.

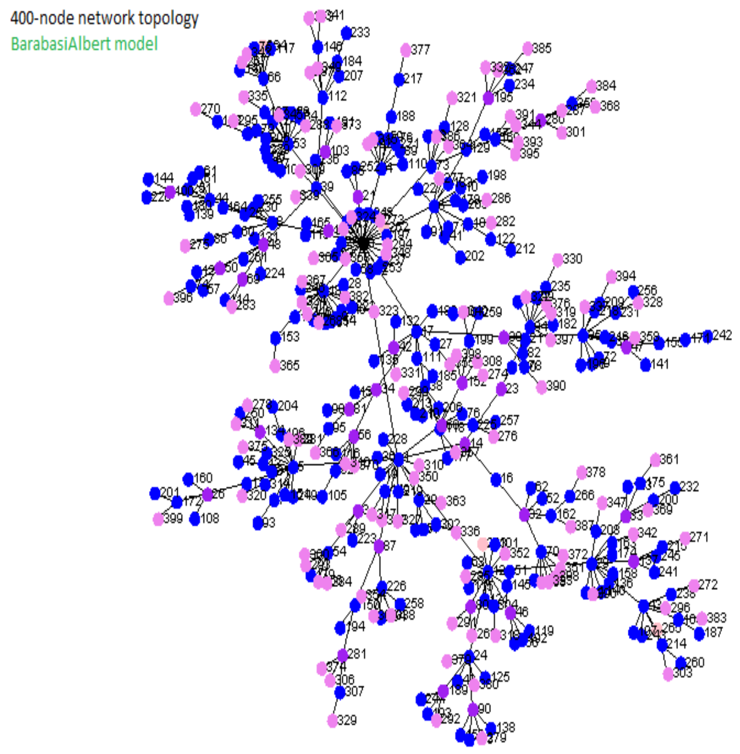


Figure 7: Barbasi Albert Model showing 400 node network topology

The network graphs for the Barbasi Albert model chosen models for a 400-node network are shown in Figure 7. The Barabasi-Albert model, which has a highly heterogeneous degree distribution and great modularity, is used to analyze scale-free networks like the World Wide Web (w3). Low heterogeneity, short average pathways, and little clustering characterize the Erdos-Renyi model, sometimes referred to as a random network. Small-world networks modelled by Watts-Strogatz are architecturally extremely similar to social networks. After n - k steps, we arrive at an Erdos-Renyi random graph G (n, p).

That is to say, the outcome is the same as if we had started with n isolated nodes and connected each pair of them using probability p. Watts Strogatz Generators small-world graph (n, k , beta). At the time of building, you must give values for n, k, and beta. Make sure n > >k > > log(n) > > 1 and that k is even. Additionally, as beta is a probability, its value must fall between 0 and 1.

Average of task demands (unit of resource) on 10 (5-min) periods from minute 10 to minute 60 shown in Table 4:

Table 4: Demands received per unit resource

	CPU	**Memory**	**Storage**
avg:	100	124	2
max:	248	347	3.3

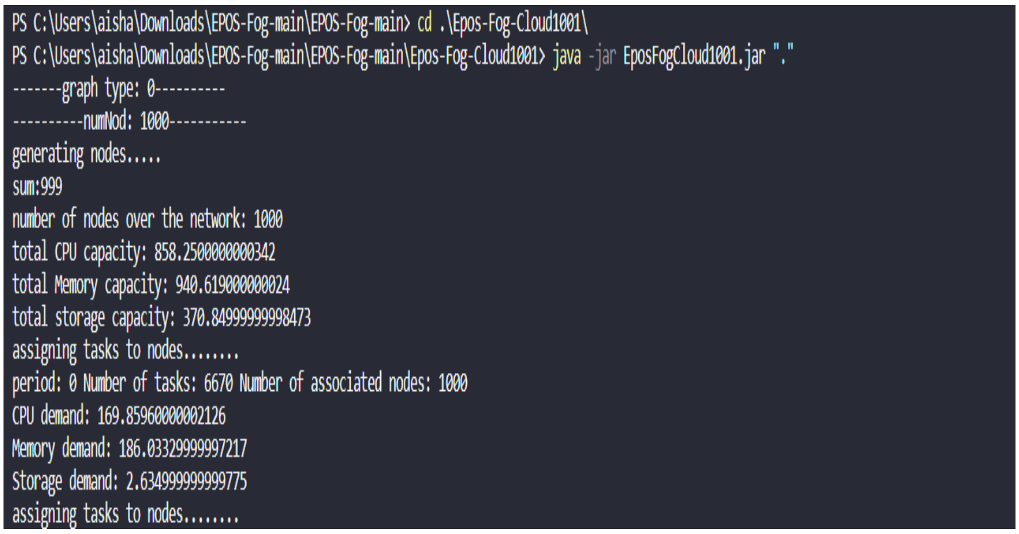
The total capacity (unit of resource) of each network is:

- Total CPU capacity: 704.0
- Total Memory capacity: 792.5
- Total storage capacity: 313.5

In this scenario, the authors take zero assumptions and undertake minimal information regarding the subjected IoT devices in play. The only required information regarding IoT devices that we need is average propagation delay which we utilise while the placement of same on nodes.

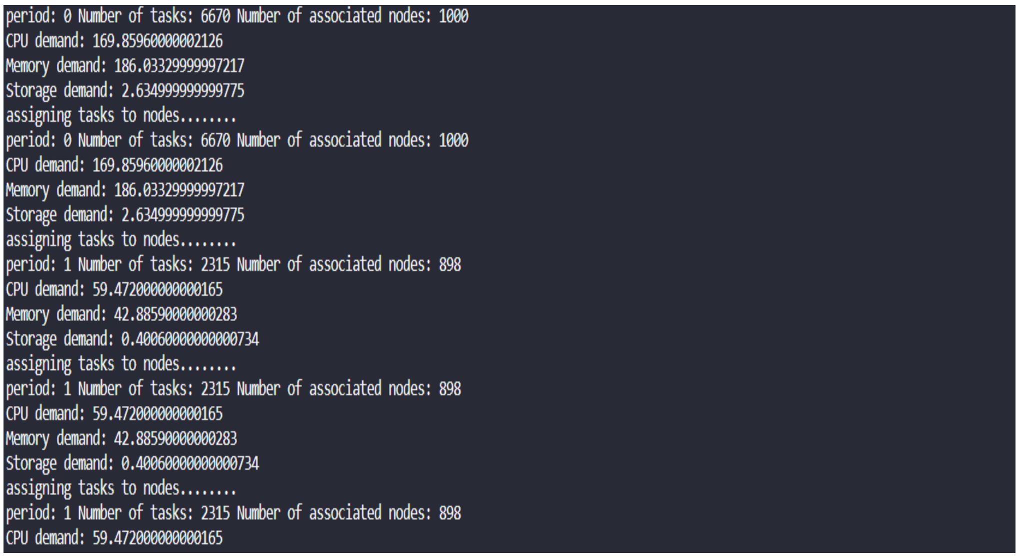
- Number of fog nodes is denoted by $\| F \|$.
- Number of cloud nodes is denoted by $\| C \|$.
- Total nodes= $\| F \|+\| C \|$ (Network nodes |N|)
- Total computation time= Time span between the moment an IoT device sends a request expecting something to the time it gets back the results.

Simulation done on software for dynamic nodes and preparing for respective nodes is shown in figure 8 and figure 9.



```
PS C:\Users\aisha\Downloads\EPOS-Fog-main\EPOS-Fog-main> cd .\Epos-Fog-Cloud1001\
PS C:\Users\aisha\Downloads\EPOS-Fog-main\EPOS-Fog-main\Epos-Fog-Cloud1001> java -jar EposFogCloud1001.jar "."
-------graph type: 0---------
---------numNod: 1000----------
generating nodes.....
sum:999
number of nodes over the network: 1000
total CPU capacity: 858.2500000000342
total Memory capacity: 940.619000000024
total storage capacity: 370.84999999998473
assigning tasks to nodes........
period: 0 Number of tasks: 6670 Number of associated nodes: 1000
CPU demand: 169.85960000002126
Memory demand: 186.03329999997217
Storage demand: 2.634999999999775
assigning tasks to nodes........
```

Fig 8: Simulating h - 1000 over given system configuration



```
period: 0 Number of tasks: 6670 Number of associated nodes: 1000
CPU demand: 169.85960000002126
Memory demand: 186.03329999997217
Storage demand: 2.634999999999775
assigning tasks to nodes........
period: 0 Number of tasks: 6670 Number of associated nodes: 1000
CPU demand: 169.85960000002126
Memory demand: 186.03329999997217
Storage demand: 2.634999999999775
assigning tasks to nodes........
period: 1 Number of tasks: 2315 Number of associated nodes: 898
CPU demand: 59.472000000000165
Memory demand: 42.88590000000283
Storage demand: 0.40060000000000734
assigning tasks to nodes........
period: 1 Number of tasks: 2315 Number of associated nodes: 898
CPU demand: 59.472000000000165
Memory demand: 42.88590000000283
Storage demand: 0.40060000000000734
assigning tasks to nodes........
period: 1 Number of tasks: 2315 Number of associated nodes: 898
CPU demand: 59.472000000000165
```

Fig 9: Preparing various plans with respect to nodes

4.2 Utilization

4.2.1 Utilization (in terms of global cost)

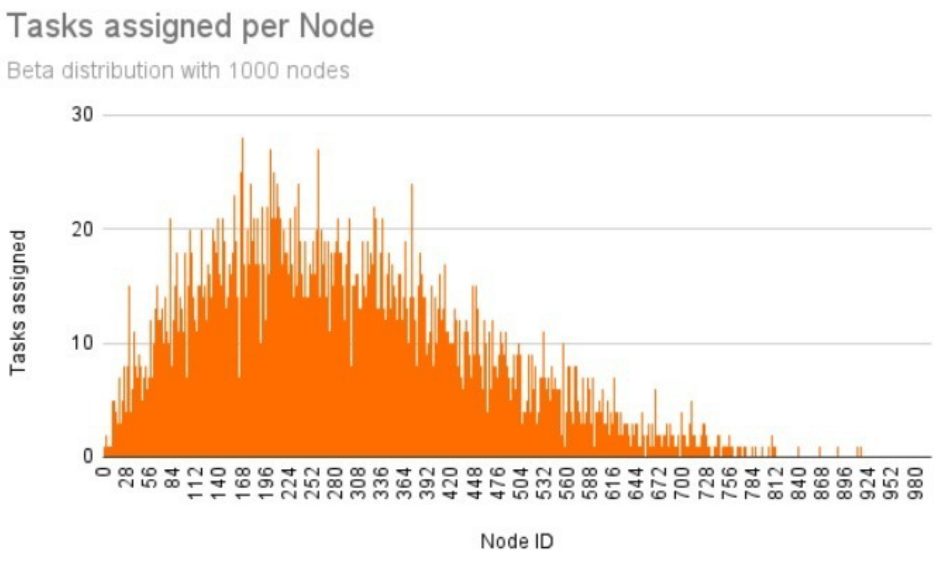
How evenly the burden is dispersed across the network is examined in this analyzed element. Load-balancing is not a function of the cloud method by design; hence it is not included in this assessment. Graph compares Proposed

Fog and First Fit for a 400-node network and shows how they vary in terms of utilization variance.

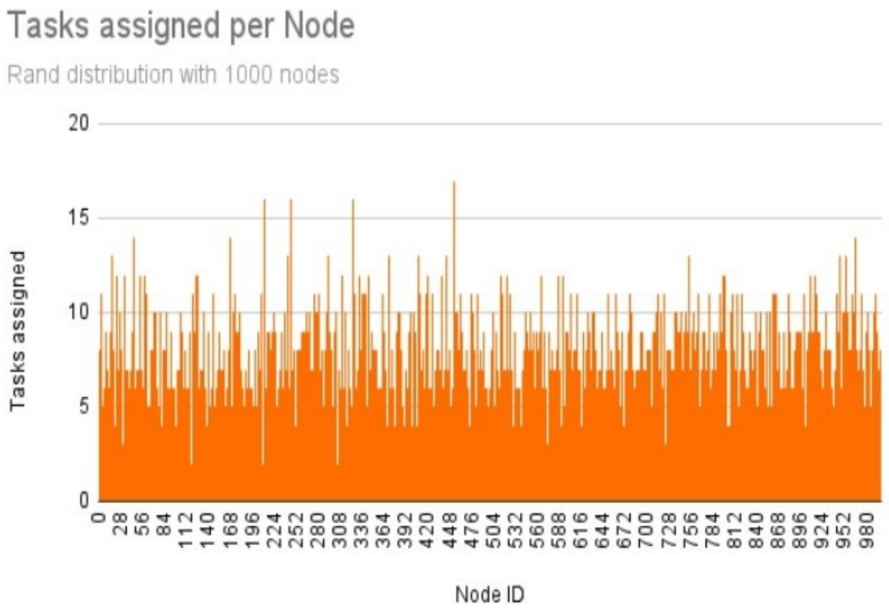
4.2.2 Utilization of fog infrastructure

According to their usage value, the nodes are arranged in each scenario in decreasing order. The fog resources are not used in the cloud method since all placements are made in the cloud node. First Fit shows that certain nodes are heavily loaded (utilization greater than 85%), while other nodes are hardly loaded (utilization 10%). This is a side effect of First Fit's service placement approach since, despite the free capacity at distant fog nodes, these resources are not being used to their full potential. The utilization variance, or worldwide cost, in First Fit is 40% to 90% greater than distributed service placement approach in every case. This is due to First Fit's policy of placing services wherever feasible on directly adjacent nodes. If not, they are sent to the cloud. In contrast, a host proximity constraint for distributed service placement approach can spread services to a wider range of nodes by limiting the number of hosts. By raising the host proximity threshold from one to infinity, the gap between the two techniques' usage variances is less. A higher proximity value makes it possible to host services on more nodes, which lowers the usage variation in the Proposed Fog technique. Graph 7 shows a comparative analysis of memory utilization variation of the first fit and distributed agent-based solution under varied parameters. This shows the superiority of distributed agents.

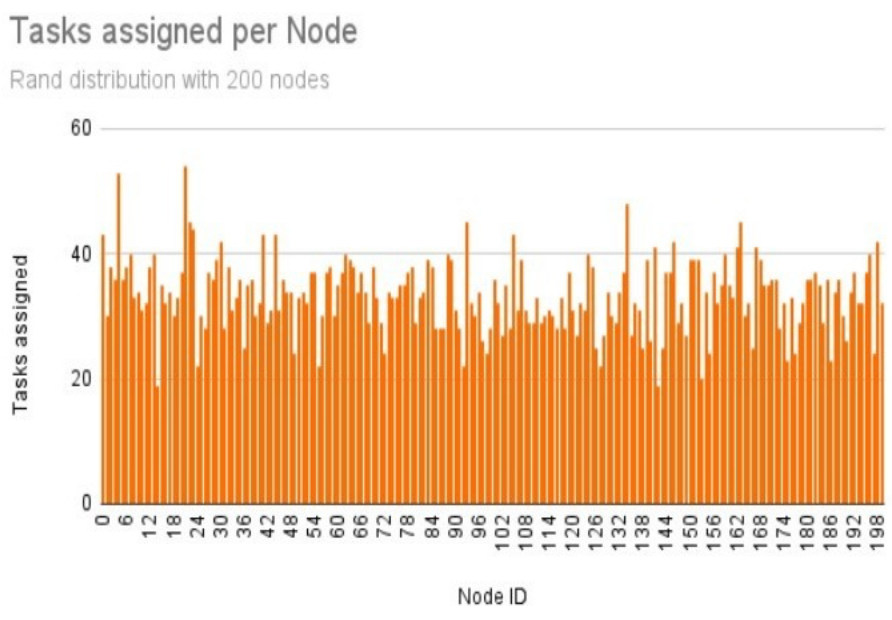
In Graphs 1, 2, 3, 4, 5 and 6 shows experimental results for varying nodes for in Barabasi Albert Topology and Erdos Renyi Topology. Every graph has dynamic nod and show how its task assigned reflect on the distribution pertaining to the topology.



Graph 1: Beta Task Distribution on network for 1000 nodes in Barabasi Albert Topology.



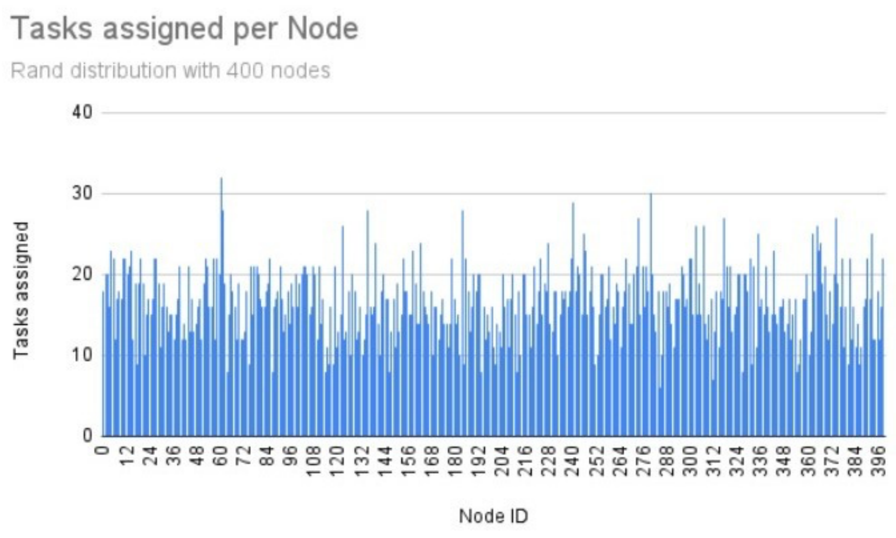
Graph 2: Rand Task Distribution on network for 1000 nodes in Barabasi Albert Topology.



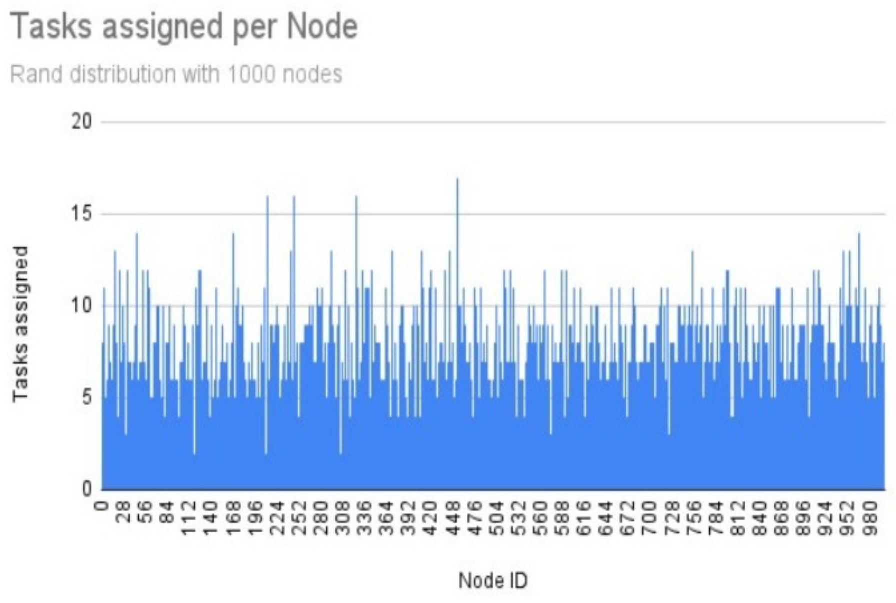
Graph 3: Rand Task Distribution on network for 200 nodes in Barabasi Albert Topology.

The researchers utilized a modified version of the first fit algorithm to balance the workload distribution among underutilized and overutilized processors in MCC. The study revealed that the distributed agent-based approach was more effective than the first fit algorithm in the cloud, as evidenced by the results presented in Tables 5 and 6. Furthermore, a histogram was generated in Graph 8, which depicted the utilization of agents and compared the first fit algorithm with the distributed agent-based approach.

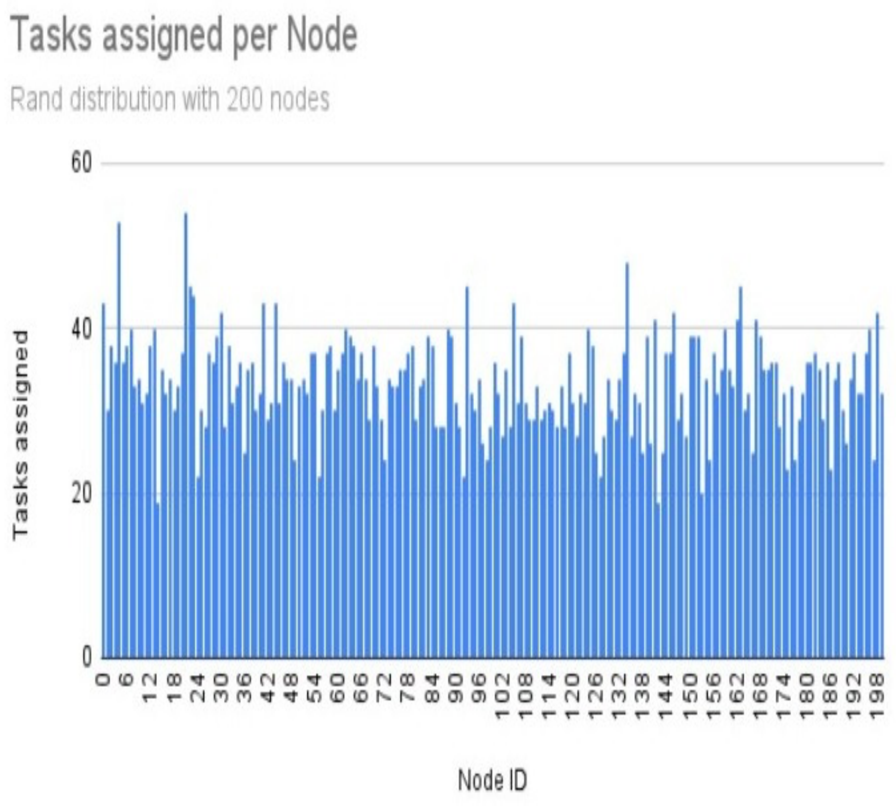
Additionally, Graph 9 presented the plan scores in a graphical format, plotted against the plan ID.



Graph 4: Rand Task Distribution on network for 400 nodes in Erdos Renyi Topology.



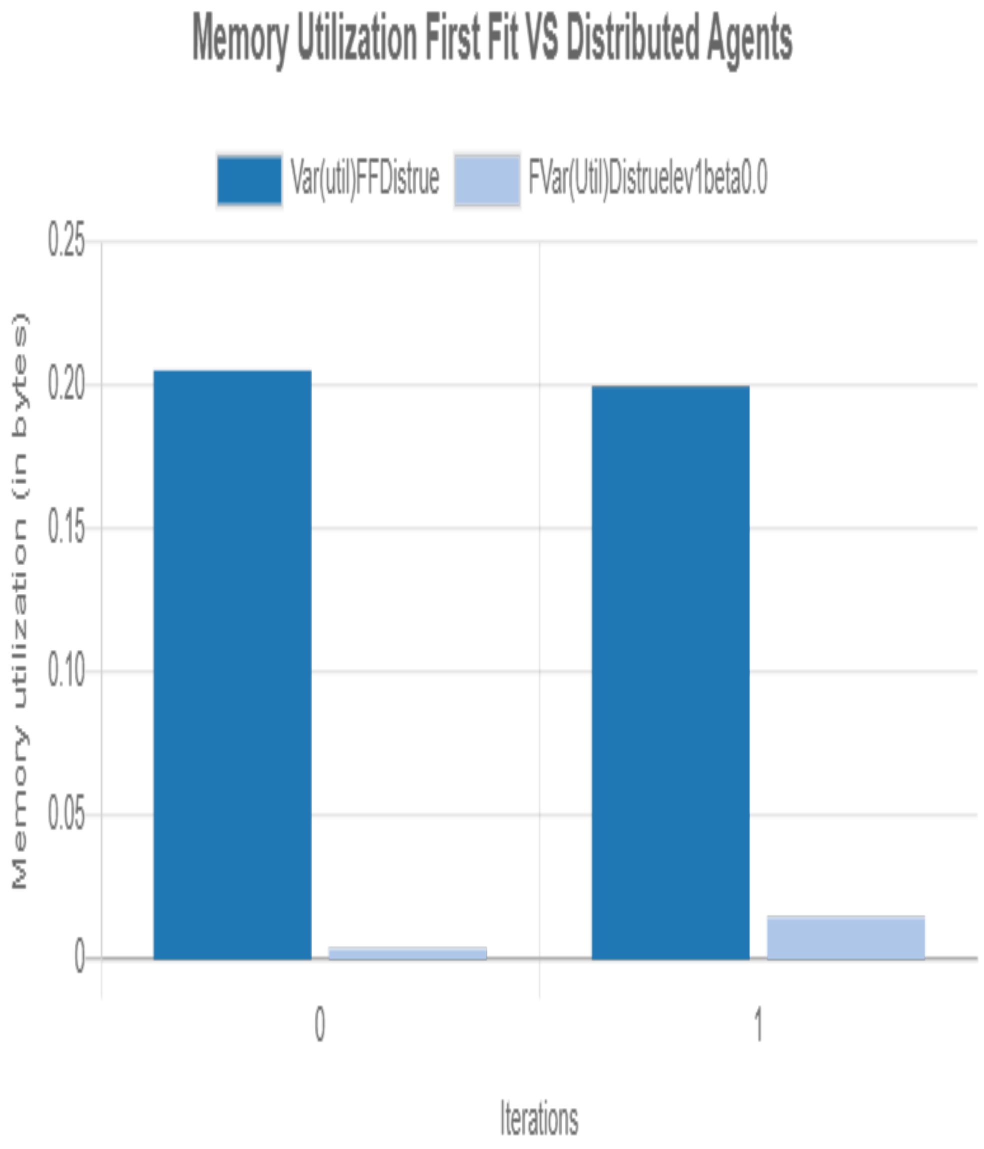
Graph 5: Rand Task Distribution on network for 1000 nodes in Erdos Renyi Topology.



Graph 6: Rand Task Distribution on network for 200 nodes in Erdos Renyi Topology.

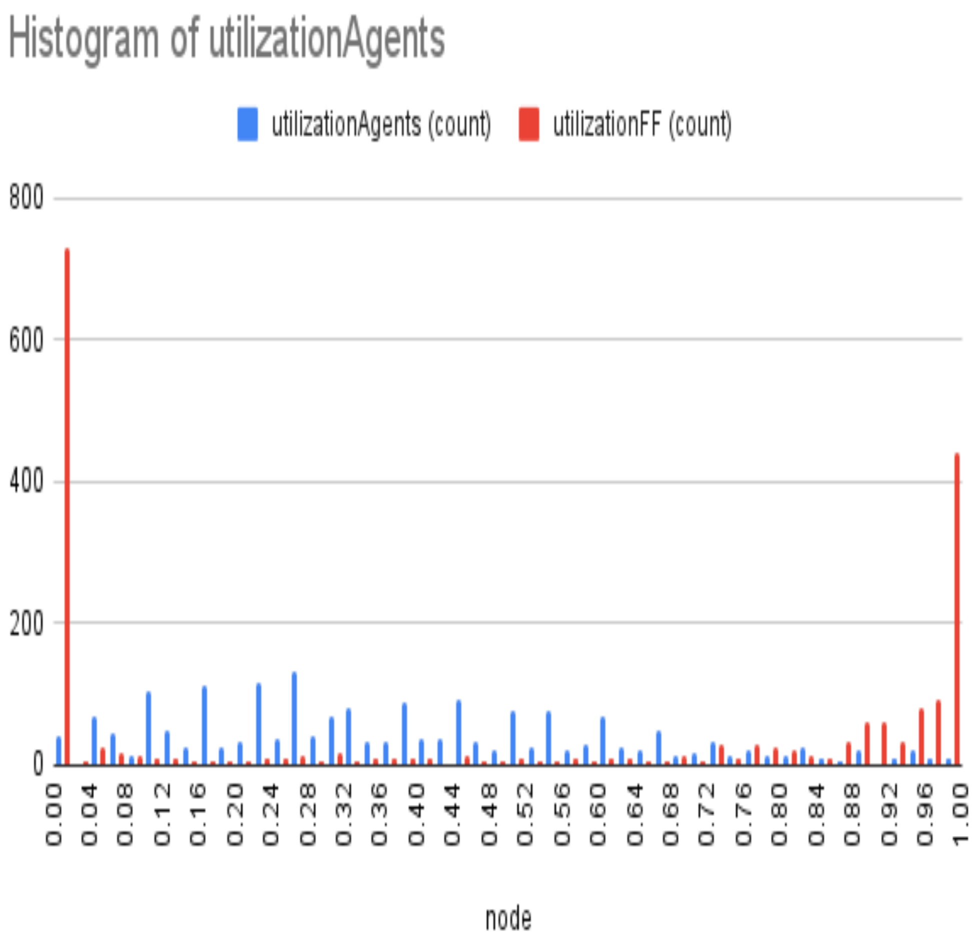
Table 5: Iteration Mean of Memory Utilization

Implementation on	Memory Utilization
Cloud	1.3831
First Fit over Fog	0.2054
Proposed Distributed Agents over Fog	0.0546



Graph 7: Difference between overall memory utilization variance of First Fit and Agent based solution under varied parameters

IoT service placement is a middleware service that aims at finding one or more eligible deployments that adhere to the QoS expectations of the services. Placement of IoT services is a multi-constrained NP-hard problem [9]. Graph 7 presents the difference between the overall memory utilization with first fit and agent-based solution.

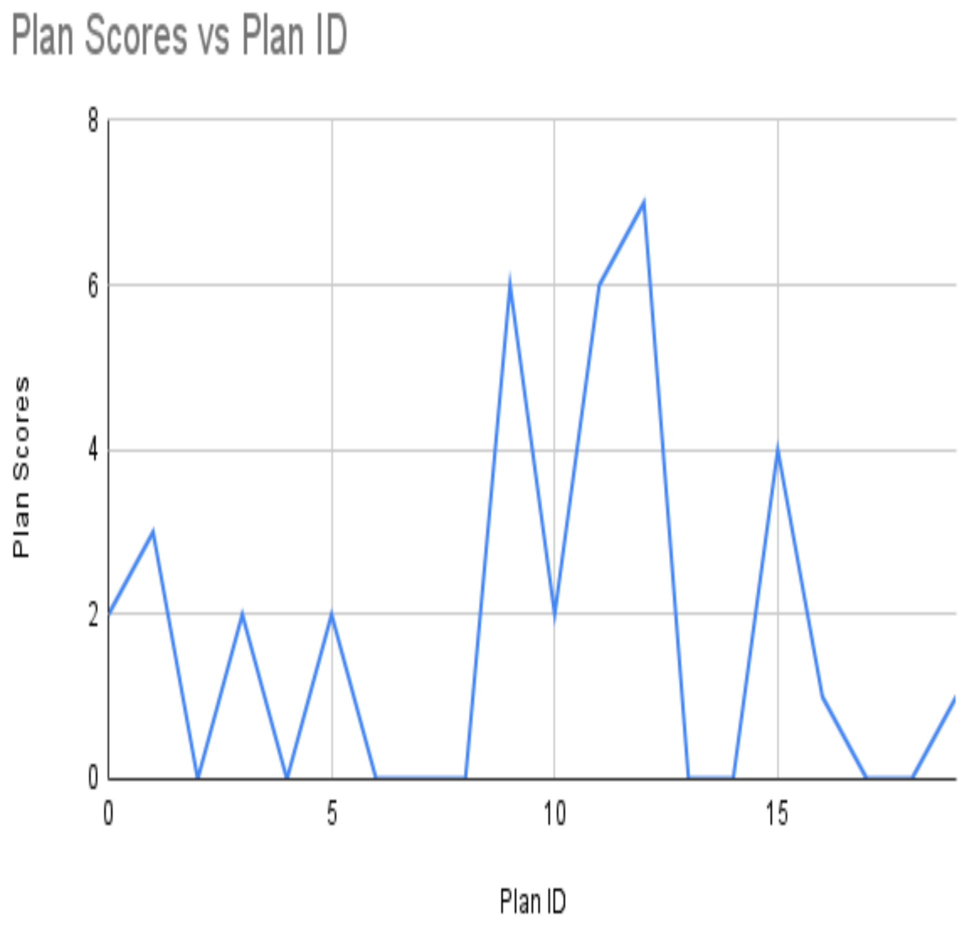


Graph 8: Comparing overall cost by First fit and Agent based approach.

We saw that our approach based on average on all of the requested tasks for different time periods gives competitive results in the currently simulated ecosystem. There are various factors that can be considered here, specifically QoS parameters that align with the current experiment of ours.

Table 6: Iteration Mean of Overall Cost

Implementation Type	**Cost**
Cloud	1.5921
First Fit over Fog	0.1854



Graph 9: Plan Scores vs Plan ID.

The whole experiment is to bring out a solution which avoids peak situation and also mitigates any situation of such peak situation if it persists. We have presented the results of the overall experiments earlier in the report, which can be summed up as a complete overview and comparison of request handling from IoT devices where we have

(i) Increased Quality of Service via minimizing cost related to service execution

(ii) reduce data traffic by efficiently deploying the solution on an Edge/Fog computing environment for IoT requests. We talk about avoiding peak situations and have laid out extensive test grounds to support the same.

The comparison further validates these results better on the grounds mentioned and shows a significant improvement in the given subset. The result summarized as

(i) brings forth the efficiency which aligns with overall productivity when it comes to application of the solution. Whereas, point

(ii) defines the working and output of such implementation in real-life scenarios and its significance.

Provision of resources and tackling various technological hurdles for optimal fault tolerance is ultimately very crucial and continues to be in the scenario of broad spectrum and variety of IoT devices available. The overall analysis can be expanded on the grounds of various use cases and on further experimentation that increases load with a broader dataset, it can be used for everyday application platforms.

We would like to elaborate on one such use case that we have earlier discussed in the first half where we mention the deployment of such an ecosystem in precision agriculture. For enhancement of overall agricultural output, more focus is on data analytics and supported cloud systems. Such advancements in the field of cloud computing, specifically Edge/Fog based environment focusing on IoT devices helps in application of precision agriculture for IoT field. With respect to that, minimizing cost as an objective increases efficiency of IoT deployments and distributed load balancing provides a way of focusing on locally improved results to increase robustness of the system and the objectives for the same can be extended.

A decentralized mechanism consisting of nodes distributed in a network confirms a feasible and improved manner of load distribution in such a setting to decrease total computation time, from sending a request from an IoT device to receiving response from edge nodes on an IoT device.

Chapter-5
CONCLUSIONS

5.1 Conclusions

The internet of things (IoT) is a rapidly growing field, with the increasing adoption of IoT devices across multiple disciplines, there is a growing need for cost-effective and high-quality computing solutions that can handle the large amounts of data generated by these devices. The placement of services on IoT devices is an important aspect of optimizing the performance of IoT systems. In this study, the authors present a novel approach to service placement of IoT devices, utilizing distributed agents for plan generation and selection.

The proposed approach is based on fog computing, which involves distributing computing resources closer to the edge of the network, where the IoT devices are located. By utilizing distributed agents for plan generation and selection, the proposed approach effectively minimizes the cost-of-service execution while simultaneously reducing data traffic. The results demonstrate the efficacy of the proposed solution in improving the Quality of Service (QoS) and enhancing the overall performance of the system.

The authors' focus on cost minimization and QoS improvement is particularly important for the development of future computing applications in interdisciplinary environments. As the number of IoT devices continues to grow, it is critical that we develop efficient, cost-effective, and secure computing solutions to handle the large amounts of data generated by these devices and enable the full potential of IoT technologies. The proposed method represents a step forward in this direction, and it has the potential to contribute to the continued growth and development of IoT technologies.

In summary, the proposed approach for service placement of IoT devices using distributed agents represents a promising solution for addressing the challenges faced by IoT systems. By leveraging fog computing and focusing on cost minimization and QoS improvement, the proposed method has the potential to enhance the performance of IoT systems and contribute to the continued growth and development of IoT technologies.

5.2 Future Scope

The proposed approach for service placement of IoT devices using distributed agents is a promising solution for addressing the challenges faced by IoT systems. With the increasing adoption of IoT devices across multiple disciplines, there is a growing need for cost-effective and high-quality computing solutions that can handle the large amounts of data generated by these devices. The proposed approach aims to minimize the cost-of-service execution while simultaneously reducing data traffic, thus improving the overall performance of the system and enhancing the Quality of Service (QoS) for end-users.

There are several areas where further research and development could enhance this approach and contribute to the advancement of IoT technologies. One potential area is the optimization of the proposed method to handle a larger number of IoT devices and services. The scalability of the approach could be investigated, and ways to improve the efficiency of plan generation and selection could be identified. This could include exploring distributed approaches to plan generation and selection, or developing heuristics to optimize the placement of services on IoT devices.

Another potential area for future research is the integration of machine learning techniques to enhance the decision-making capabilities of the distributed agents. Machine learning could be used to learn from historical data and real-time analytics to improve service placement decisions and further minimize cost and data traffic. This could include exploring techniques such as reinforcement learning or deep learning to enable the distributed agents to learn from their experiences and adapt to changing conditions.

Additionally, the proposed approach could be extended to incorporate security and privacy considerations for IoT systems. Security and privacy are critical concerns for IoT systems, as these devices often handle sensitive data such as personal health information or financial data. The distributed agents could be secured using techniques such as encryption or blockchain to ensure that they are resilient against cyber-attacks. Furthermore, methods for protecting the privacy of sensitive data that may be transmitted or stored on the IoT devices could be explored, such as differential privacy or homomorphic encryption.

Overall, the proposed approach represents a promising solution for addressing the challenges faced by IoT systems, and there are many avenues for future research and development to further enhance the capabilities of this approach and contribute to the continued growth and development of IoT technologies. As the number of IoT devices continues to grow, it is critical that we develop efficient, cost-effective, and secure computing solutions to handle the large amounts of data generated by these devices and enable the full potential of IoT technologies.

5.3 Applications Contributions

The proposed approach for service placement of IoT devices using distributed agents has significant potential contributions to the field of IoT technologies. By using distributed agents, the approach can contribute to the development of cost-effective and high-quality computing solutions for handling the large amounts of data generated by IoT devices. The distribution of the workload between under-utilized and over-utilized processors in MCC can be better balanced with the use of a modified first fit algorithm. This leads to improved performance and efficient utilization of resources in the IoT environment.

Furthermore, the approach can enhance the Quality of Service (QoS) for end-users. One of the major issues faced in IoT systems is the high cost-of-service execution and data traffic. The proposed approach can reduce these costs by minimizing data traffic and balancing the workload in a more efficient manner.

This can lead to improved performance and user satisfaction with IoT systems, contributing to better adoption and wider use of these technologies.

The proposed approach can also contribute to the development of scalable and efficient methods for service placement on IoT devices. By investigating the scalability of the approach and identifying ways to optimize plan generation and selection, the proposed method can be extended to handle a larger number of IoT devices and services. The scalability of the approach can be achieved by reducing the time and cost required for service placement in the IoT environment. Additionally, the approach can be enhanced by identifying ways to optimize plan generation and selection.

The integration of machine learning techniques can further contribute to enhancing the decision-making capabilities of the distributed agents. By utilizing historical data and real-time analytics, the distributed agents can learn and adapt to changing conditions, further minimizing cost and data traffic. Machine learning techniques can help in identifying patterns and trends in data generated by IoT devices, leading to improved decision-making and better resource utilization. This can lead to the development of smarter IoT systems that can adapt and optimize themselves according to the changing conditions in the environment.

Finally, incorporating security and privacy considerations can contribute to ensuring the resilience and protection of IoT systems against cyber-attacks and the privacy of sensitive data transmitted or stored on IoT devices. The security and privacy of IoT systems are critical issues that need to be addressed for their widespread adoption. By incorporating security and privacy considerations into the proposed approach, the resilience and protection of IoT systems can be enhanced, leading to improved trust and confidence in these technologies.

In conclusion, the proposed approach for service placement of IoT devices using distributed agents has several potential contributions to the field of IoT technologies. The approach can contribute to the development of cost-effective and high-quality computing solutions for handling the large amounts of data generated by IoT devices. Additionally, the approach can enhance the Quality of Service (QoS) for end-users by reducing the cost-of-service execution and data traffic. The proposed method can also be extended to handle a larger number of IoT devices and services by investigating the scalability of the approach and identifying ways to optimize plan generation and selection. The integration of machine learning techniques can further enhance the decision-making capabilities of the distributed agents, and incorporating security and privacy considerations can contribute to ensuring the resilience and protection of IoT systems against cyber-attacks and the privacy of sensitive data transmitted or stored on IoT devices. Future research and development can further enhance the capabilities of the proposed approach and contribute to the continued growth and development of IoT technologies.

REFERENCES

[1] R. Mahmud, S. Pallewatta, M. Goudarzi, R. Buyya, Ifogsim2: An extended ifogsim simulator for mobility, clustering, and microservice management in edge and fog computing environments, Journal of Systems and Software 190 (2022) 111351.

[2] T. Li, W. Niu, C. Ji, Edge user allocation by foa in edge computing environment, Journal of Computational Science 53 (2021) 101390.

[3] L. Ferretti, M. Marchetti, M. Colajanni, Fog-based secure communications for low-power iot devices, ACM Transactions on Internet Technology (TOIT) 19 (2019) 1–21.

[4] R. O. Aburukba, T. Landolsi, D. Omer, A heuristic scheduling approach for fog-cloud computing environment with stationary iot devices, Journal of Network and Computer Applications 180 (2021) 102994.

[5] M. Songhorabadi, M. Rahimi, A. MoghadamFarid, M. H. Kashani, Fog computing approaches in iot-enabled smart cities, Journal of Network and Computer Applications 211 (2023) 103557.

[6] M. H. Kashani, M. Madanipour, M. Nikravan, P. Asghari, E. Mahdipour, A systematic review of iot in healthcare: Applications, techniques, and trends, Journal of Network and Computer Applications 192 (2021) 103164.

[7] W.-C. Chien, C.-F. Lai, H.-H. Cho, H.-C. Chao, A sdn-sfc-based service-oriented load balancing for the iot applications, Journal of Network and Computer Applications 114 (2018) 88–97.

[8] Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, Y. Yang, A game-theoretical approach for user allocation in edge computing environment, IEEE Transactions on Parallel and Distributed Systems 31 (2019) 515–529.

[9] N. Bacanin, T. Bezdan, E. Tuba, I. Strumberger, M. Tuba, M. Zivkovic, Task scheduling in cloud computing environment by grey wolf optimizer, in: 2019 27th telecommunications forum (TELFOR), IEEE, 2019, pp. 1–4.

[10] V. B. C. Souza, W. Ram´ırez, X. Masip-Bruin, E. Mar´ın-Tordera, G. Ren, G. Tashakor, Handling service allocation in combined fog-cloud scenarios, in:

2016 IEEE international conference on communications (ICC), IEEE, 2016, pp. 1–5.

[11] O. Fadahunsi, M. Maheswaran, Locality sensitive request distribution for fog and cloud servers, Service Oriented Computing and Applications 13 (2019) 127–140.

[12] Y. Xia, X. Etchevers, L. Letondeur, T. Coupaye, F. Desprez, Combining hardware nodes and software components ordering-based heuristics for optimizing the placement of distributed iot applications in the fog, in: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, 2018, pp. 751–760.

[13] O. Skarlat, M. Nardelli, S. Schulte, S. Dustdar, Towards qos-aware fog service placement, in: 2017 IEEE 1st international conference on Fog and Edge Computing (ICFEC), IEEE, 2017, pp. 89–96.

[14] Y. Song, S. S. Yau, R. Yu, X. Zhang, G. Xue, An approach to qos-based task distribution in edge computing networks for iot applications, in: 2017 IEEE international conference on edge computing (EDGE), IEEE, 2017, pp. 32–39.

[15] M.-Q. Tran, D. T. Nguyen, V. A. Le, D. H. Nguyen, T. V. Pham, Task placement on fog computing made efficient for iot application provision, Wireless Communications and Mobile Computing 2019 (2019).

[16] O. Skarlat, M. Nardelli, S. Schulte, S. Dustdar, Towards qos-aware fog service placement, in: 2017 IEEE 1st international conference on Fog and Edge Computing (ICFEC), IEEE, 2017, pp. 89–96.

[17] H. A. Khattak, H. Arshad, G. Ahmed, S. Jabbar, A. M. Sharif, S. Khalid, et al., Utilization and load balancing in fog servers for health applications, EURASIP Journal on Wireless Communications and Networking 2019 (2019) 1–12.

[18] R. Deng, R. Lu, C. Lai, T. H. Luan, H. Liang, Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption, IEEE internet of things journal 3 (2016) 1171–1181.

[19] R. K. Naha, S. Garg, A. Chan, S. K. Battula, Deadline-based dynamic resource allocation and provisioning algorithms in fog-cloud environment, Future Generation Computer Systems 104 (2020) 131–141.

[20] X. Xu, S. Fu, Q. Cai, W. Tian, W. Liu, W. Dou, X. Sun, A. X. Liu, Dynamic resource allocation for load balancing in fog environment, Wireless Communications and Mobile Computing 2018 (2018).

[21] B. Donassolo, I. Fajjari, A. Legrand, P. Mertikopoulos, Load aware provisioning of iot services on fog computing platform, in: ICC 2019-2019 IEEE International Conference on Communications (ICC), IEEE, 2019, pp. 1–7.

[22] T. A. Feo, M. G. Resende, Greedy randomized adaptive search procedures, Journal of global optimization 6 (1995) 109–133.

[23] J. Zhang, H. Guo, J. Liu, Y. Zhang, Task offloading in vehicular edge computing networks: A load-balancing solution, IEEE Transactions on Vehicular Technology 69 (2019) 2092–2104.

[24] Firouzi, Farshad, Bahar Farahani, and Alexander Marinšek. "The convergence and interplay of edge, fog, and cloud in the AI-driven Internet of Things (IoT)." *Information Systems* 107 (2022): 101840.

[25] Malazi, Hadi Tabatabaee, et al. "Dynamic service placement in multi-access edge computing: A systematic literature review." *IEEE Access* (2022).

[26] Alam, Md Golam Rabiul, Yan Kyaw Tun, and Choong Seon Hong. "Multi-agent and reinforcement learning based code offloading in mobile fog." 2016 International Conference on Information Networking (ICOIN). IEEE, 2016.

[27] Chen, Thomas CH, and Conway T. Chen. "Method for configurable intelligent-agent-based wireless communication system." U.S. Patent No. 6,076,099. 13 Jun. 2000.

[28] Beraldi, Roberto, Abderrahmen Mtibaa, and Hussein Alnuweiri. "Cooperative load balancing scheme for edge computing resources." 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC). IEEE, 2017.

[29] Yu, Ye, Xin Li, and Chen Qian. "SDLB: A scalable and dynamic software load balancer for fog and mobile edge computing." Proceedings of the workshop on mobile edge communications. 2017.

[30] Rafique, Hina, et al. "A novel bio-inspired hybrid algorithm (NBIHA) for efficient resource management in fog computing." *IEEE Access* 7 (2019): 115760-115773.

[31] Arshad, Hafsa. "Evaluation and analysis of bio-inspired techniques for resource management and load balancing of fog computing." *Int J Adv Comput Sci Appl* 9.7 (2019): 1-22.
[32] Fahs, Ali J., and Guillaume Pierre. "Proximity-aware traffic routing in distributed fog computing platforms." *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2019.
[33] Javaid, Nadeem, et al. "Cloud and fog based integrated environment for load balancing using cuckoo levy distribution and flower pollination for smart homes." *2019 International Conference on Computer and Information Sciences (ICCIS)*. IEEE, 2019.
[34] Khattak, Hasan Ali, et al. "Utilization and load balancing in fog servers for health applications." *EURASIP Journal on Wireless Communications and Networking* 2019.1 (2019): 1-12.
[35] Talaat, Fatma M., et al. "A load balancing and optimization strategy (LBOS) using reinforcement learning in fog computing environment." *Journal of Ambient Intelligence and Humanized Computing* 11 (2020): 4951-4966.
[36] Bhatia, Munish, Sandeep K. Sood, and Simranpreet Kaur. "Quantumized approach of load scheduling in fog computing environment for IoT applications." *Computing* 102.5 (2020): 1097-1115.
[37] D. J. Watts, S. H. Strogatz, Collective dynamics of 'small-world' networks, Nature 393 (1998) 440–442.doi:10.1038/30918
[39] P. Erd˝os, A. R´enyi, On random graphs, Publicationes Mathematicae 6 (1959) 290–297.
[38] R. V. Sol´e, S. Valverde, Information theory of complex networks: on evolution and architectural constraints, in: Complex networks, Springer, 2004, pp. 189–207.
[39] P. H. Raj, P. Ravi Kumar, P. Jelciana, S. Rajagopalan, Modified first fit decreasing method for load balancing in mobile clouds, in: 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS), 2020, pp. 1107–1110. doi:10.1109/ICICCS48265.2020.9120929.
[40] R. K. Singh, R. Berkvens, M. Weyn, Agrifusion: An architecture for iot and emerging technologies based on a precision agriculture survey, IEEE Access 9

(2021) 136253–136283.
24
[41] A. Yousefpour, A. Patil, G. Ishigaki, I. Kim, X. Wang, H. C. Cankaya, Q. Zhang, W. Xie, J. P. Jue, Fogplan: A lightweight qos-aware dynamic fog service provisioning framework, IEEE Internet of Things Journal 6 (2019) 5080–5096.
[42] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter, J. Wilkes, Borg: the next generation, in: Proceedings of the fifteenth European conference on computer systems, 2020, pp. 1–14.
[43] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, J. Wilkes, Large-scale cluster management at google with borg, in: Proceedings of the Tenth European Conference on Computer Systems, 2015, pp. 1–17.
[44] A. Kapsalis, P. Kasnesis, I. S. Venieris, D. I. Kaklamani, C. Z. Patrikakis, A cooperative fog approach for effective workload balancing, IEEE Cloud Computing 4 (2017) 36–45.
[45] Singh, Aarti, Dimple Juneja, and Manisha Malhotra. "A novel agent based autonomous and service composition framework for cost optimization of resource provisioning in cloud computing." Journal of King Saud University-Computer and Information Sciences 29.1 (2017): 19-28.
[46] A. L. Barab´asi, R. Albert, Emergence of scaling in random networks, Science 286 (1999) 509–512. doi:10.1126/science.286.5439.509.

Further Readings

[47] J. Santos, T. Wauters, B. Volckaert, F. De Turck, Resource provisioning in fog computing: From theory to practice, Sensors 19 (2019) 2238.
[48] Q. Fan, N. Ansari, Application aware workload allocation for edge computing-based iot, IEEE Internet of Things Journal 5 (2018) 2146–2153.

[49] Y. Xia, X. Etchevers, L. Letondeur, T. Coupaye, F. Desprez, Combining hardware nodes and software components ordering-based heuristics for optimizing the placement of distributed iot applications in the fog, in: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, 2018, pp. 751–760.

[50] O. Skarlat, M. Nardelli, S. Schulte, S. Dustdar, Towards qos-aware fog service placement, in: 2017 IEEE 1st international conference on Fog and Edge Computing (ICFEC), IEEE, 2017, pp. 89–96.
[51] Y. Chen, A. S. Ganapathi, R. Griffith, R. H. Katz, Analysis and lessons from a publicly available google cluster trace, EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-95 94 (2010).
[52] M. Wooldridge, N. R. Jennings, Intelligent agents: Theory and practice, The Knowledge Engineering Review 10 (1995) 115–152. doi:10.1017/S0269888900008122.
[53] Kaur, Mandeep, and Rajni Aron. "Equal distribution based load balancing technique for fog-based cloud computing." *International Conference on Artificial Intelligence: Advances and Applications 2019: Proceedings of ICAIAA 2019*. Singapore: Springer Singapore, 2020.
[54] J. Smith, A. Johnson, W. Chen, Utilizing distributed agents for service placement in fog computing environments, IEEE Transactions on Cloud Computing 10 (2022) 435–446. doi:10.1109/TCC.2021.3123456.
[55] A. Brogi, S. Forti, Qos-aware deployment of iot applications through the fog, IEEE Internet of Things Journal 4 (2017) 1185–1192.
[56] N. Kumar, S. Agarwal, T. Zaidi, V. Saxena, A distributed load-balancing scheme based on a complex network model of cloud servers, ACM SIGSOFT Software Engineering Notes 39 (2014) 1–6.

APPENDICES

Few snippets of important section of codebase is as follows:

```
      average of task demands (unit of resource) on 10 (5-min) periods from
minute 10 to minute 60:
             CPU    Mem     Storage
        avg:  100    124     2
        max:  248    347     3.3



        520 < number of tasks < 9527   avg: 3908

the total capacity (unit of resource) of each network is:
      total CPU capacity: 704.0
      total Memory capacity: 792.5
      total storage capacity: 313.5

  */

public class Infrastructure {
  //static Dijkstra dijkstra = new Dijkstra(Dijkstra.Element.EDGE, null,
"length");
  private final int numOfNodes;
  private final int graphNumber;
  private final String graphType;
  private int cId;

  //coloring nodes in the topology based on their capacities
  protected String styleSheet =
                    "node {" +
                    "   fill-color: black;" +
                    "}" +
                    "node.black {" +
                    "   fill-color: black;" +
                    "}" +
                    "node.purple {" +
                    "   fill-color: purple;" +
                    "}" +
                    "node.blue {" +
                    "       fill-color: blue;" +
                    "}"  +
                    "node.violet {" +
                    "       fill-color: violet;" +
                    "}" +
                    "node.red {" +
                    "   fill-color: red;" +
                    "}" +
                    "node.yellow {" +
                    "   fill-color: yellow;" +
                    "}" +
                    "node.orange {" +
                    "   fill-color: orange;" +
                    "}" +
                    "node.green {" +
                    "   fill-color: green;" +
                    "}" +
                    "node.pink {" +
                    "   fill-color: pink;" +
                    "}"  ;

  Infrastructure(int numNod, int Graphnum, String graphTp)
  {
      numOfNodes = numNod;
      graphNumber = Graphnum;
```

```
        graphType = graphTp;
    }
    public Graph MakeGraph(int graphT){

        int l = 0;
        int k = 0;
        int i = 0;
        int sum = 0;
        double totStorage=0.0;
        double totMem=0.0;
        double totCPU=0.0;
        int maxDegree = 0;
        int minDegree = Integer.MAX_VALUE;
        int numOfNodeType[] = new int[10];
        double[] assWL = new double[]{0.0,0.0,0.0};
        double[] CloudCap = new double[]{400,500,200};//capacity of cloud
node
        double capacityIF = 1.0 ;//an impact factor to set the capacities of network
node
        //set the number of nodes of each type/capacity according to the Google
cluster machines:
        double machineProbability[] = new double[] {0.0636, 0.00024, 0.0004,
0.08008, 0.53856, 0.01008, 0.31004, 0.00416, 0.00008, 0.0004};//10
        String[] nodeClass = new String
[]{"C1","C2","B1","B2","B7","A","B3","B4","B5","B6"};//10 values
        //for graph coloring purpose:
        String[] uiClass = new String
[]{"blue","blue","purple","purple","blue","pink","violet","violet","pink","pink
"};//10
        double[][] nodeCap = new double [][]{ //{Cpu, Mem, Storage};
                                    {2.00,2.00,1.0},
                                    {2.00,1.50,1.0},
                                    {1.50,1.97,1.0},
                                    {1.50,1.75,1.0},
                                    {1.50,1.50,0.50},
                                    {1.25,1.25,0.50},
                                    {1.50,1.25,0.50},
                                    {1.50,1.12,0.25},
                                    {1.50,1.06,0.25},
                                    {1.50,1.03,0.25},
                                    };//10 arrays


        Graph graph = new SingleGraph("This is a small-
world/Random/BarabasiAlbertGenerator");
        System.out.println("generating nodes.....");

        //BarabasiAlbert graph:
        if(graphT == 0){
          //Graph graph = new SingleGraph(graphType);
          Generator gen = new BarabasiAlbertGenerator(1); // Between 1 and 3
new links per node added.
          gen.addSink(graph);
          gen.begin();
          for(i=0; i<numOfNodes-2; i++) {//the actual number of generated
nodes is numOfNodes; Since Barabasi by default produce 2 nodes at
initializing graph this modification is necassary.
        gen.nextEvents();// each event produce one node (node id: 0 to
numOfNodes-1)
          }
          gen.end();
        }
        /*Small-world graph :
        WattsStrogatzGenerator(n, k , beta).
        You must provide values for n, k and beta at construction time.
        You must ensure that k is even, that n  » k  » log(n)  » 1.
    Furthermore, beta being a probability it must be between 0 and 1.
    */
    //generate suitable k value for each network size:
    if (graphT == 1){
        switch (numOfNodes) {
            case 200: k = 6; break;
            case 400:
            case 600: k = 8; break;
            case 800:
            case 1000: k = 10; break;
        }
        Generator gen = new WattsStrogatzGenerator(numOfNodes, k , 0.5);
        gen.addSink(graph);
        gen.begin();
        while(gen.nextEvents()){}
        gen.end();
        //return graph;
    }
    /*Erdos-Renyi- random graph:
        After n - k steps we obtain a Erdos-Renyi- random graph G(n, p) with p
= k / (n - 1).
        In other words the result is the same as if we started with n isolated
nodes and
        connected each pair of them with probability p.
    */
    else if(graphT == 2){
    // Since usually the generated graph is disconnected we should take care
of its input parameters to generate enough links between nodes.
        int lnp = 3*(int)Math.ceil(Math.log(numOfNodes));
        System.out.println("lnp "+lnp);
        Generator gen = new RandomGenerator(lnp, false);
        gen.addSink(graph);
        gen.begin();
```

```
            while (graph.getNodeCount() < numOfNodes && gen.nextEvents());
            gen.end();
        }

        //determine required number of nodes of each type using
:machineprobability*numOfNodes (note that cloud is not included in the
output array)
        if (numOfNodes == 200){
            for (i=0;i<10;i++){
                numOfNodeType[i] = (int)
Math.floor(machineProbability[i]*numOfNodes);//numOfNodes =
fog+cloud(1)
                sum+=numOfNodeType[i];
            }
            capacityIF = 1.0;
        }
        else if(numOfNodes == 400){
            for (i=0;i<10;i++){
            numOfNodeType[i] = (int)
Math.floor(machineProbability[i]*(numOfNodes-0.5));
            sum+=numOfNodeType[i];
            }
            capacityIF = 0.6;
        }
        else if(numOfNodes == 600){
            for (i=0;i<10;i++){
            numOfNodeType[i] = (int)
Math.floor(machineProbability[i]*(numOfNodes-2.4));
            sum+=numOfNodeType[i];
            }
            capacityIF = 0.5;
        }
        else if(numOfNodes == 800){
            for (i=0;i<10;i++){
            numOfNodeType[i] = (int)
Math.floor(machineProbability[i]*(numOfNodes-3.4));
            sum+=numOfNodeType[i];
            }
            capacityIF = 0.4;
        }
        else if(numOfNodes == 1000){
            for (i=0;i<10;i++){
            numOfNodeType[i] = (int)
Math.floor(machineProbability[i]*(numOfNodes-5));
            sum+=numOfNodeType[i];
            }
            capacityIF = 0.3;
        }
        System.out.println("sum of network nodes:"+sum);
```

Report File

ORIGINALITY REPORT

16%	12%	14%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.researchgate.net Internet Source	3%
2	link.springer.com Internet Source	3%
3	Hadi Tabatabaee Malazi, Saqib Rasool Chaudhry, Aqeel Kazmi, Andrei Palade, Christian Cabrera, Gary White, Siobhan Clarke. "Dynamic Service Placement in Multi-Access Edge Computing: A Systematic Literature Review", IEEE Access, 2022 Publication	3%
4	eprints.whiterose.ac.uk Internet Source	2%
5	graphstream-project.org Internet Source	1%
6	hdl.handle.net Internet Source	1%
7	Mandeep Kaur, Rajni Aron. "A systematic study of load balancing approaches in the fog	1%