DESIGN AND DEVELOPMENT OF ENERGY EFFICIENCY BASED RESOURCE ALLOCATION FRAMEWORK FOR FOG COMPUTING ENVIRONMENT

Thesis Submitted for the Award of the Degree of

DOCTOR OF PHILOSOPHY

in

Computer Application

By

Satyakam Rahul

Registration Number: 11919193

Supervised By

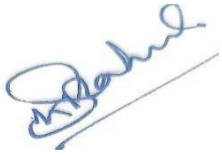
Dr. Vinay Bhardwaj (23825)

Department of Computer Science &Engineering (Assistant Professor)
Lovely Professional University

LOVELY PROFESSIONAL UNIVERSITY
PUNJAB
2025

DECLARATION

I, hereby declared that the presented work in the thesis entitled "(DESIGN AND DEVELOPMENT OF ENERGY EFFICIENCY BASED RESOURCE ALLOCATION FRAMEWORK FOR FOG COMPUTING ENVIRONMENT" in fulfilment of degree of **Doctor of Philosophy (Ph. D.)** is outcome of research work carried out by me under the supervision of Dr. Vinay Bhardwaj, working as Assistant Professor, in the Computer Science Engineering, School of Computing of Lovely Professional University, Punjab, India. In keeping with general practice of reporting scientific observations, due acknowledgements have been made whenever work described here has been based on findings of other investigator. This work has not been submitted in part or full to any other University or Institute for the award of any degree.



(Signature of Scholar)

Name of the scholar: Satyakam Rahul

Registration No. : 11919193

Department/school: Computer Application

Lovely Professional University,

Punjab, India

CERTIFICATE

This is to certify that the work reported in the Ph. D. thesis entitled DESIGN AND DEVELOPMENT OF ENERGY EFFICIENCY BASED RESOURCE ALLOCATION FRAMEWORK FOR FOG COMPUTING ENVIRONMENT submitted in fulfillment of the requirement for the award of degree of **Doctor of Philosophy (Ph.D.)** in the _Computer Applications/ School of Computing, is a research work carried out by Satyakam Rahul, 11919193, is bonafide record of his/her original work carried out under my supervision and that no part of thesis has been submitted for any other degree, diploma or equivalent course.

(Signature of Supervisor)	**(Signature of Co-Supervisor)**
Name of supervisor: Dr. Vinay Bhardwaj	Name of Co-Supervisor:
Designation: Assistant Professor	Designation:
Department/school: Computer Science Engineering School of computing	Department/school:
University: Lovely Professional University, Punjab	University

TABLE OF CONTENTS

List of Tables

List of Figures

List of Abbreviations

IoT Internet of Things

IoE Internet of Everything

RMS Resource Management System

GWO Grey Wolf Optimization

PSO Particle Swarn Optimization

ACO Ant Colony Optimization

SAA Simulated Annealing Approach

DAG Directed Acyclic Graphs

LIGO Laser Interferometer Gravitational Wave Observator

Acknowledgement:

I am grateful to all those who have contributed to the completion of this PhD thesis. I would like to extend my sincere appreciation to Dr. Vinay Bhardwaj for his invaluable guidance throughout this research endeavour. My supervisor has been a constant source of knowledge, inspiration, motivation, and encouragement during the entire duration of this research work. I would also like to acknowledge the management of Lovely Professional University for their unwavering support and assistance in enabling me to balance my work and research commitments. The doctoral program at LPU has made it possible for me to pursue my academic aspirations and enhance my knowledge. Special thanks to the examiners of end-term reports and the reviewers of journals for their insightful feedback that has helped in enhancing the quality of my work.

I am deeply grateful to all my teachers who have played a significant role in shaping my academic journey and skill development since my formative years. My heartfelt thanks to my parents and my family members for their love, support, and unwavering belief in my abilities, which has been a constant source of strength in achieving my life goals. I am thankful to my wife, Sharda, for her unending support throughout my research endeavour.

I would like to express my gratitude to my senior Dr. Mandeep Kaur for her support and guidance, as well as my friends for their continued encouragement. Lastly, I extend my thanks to every individual who has provided direct or indirect assistance and motivation during this challenging task.

Abstract

Fog computing is increasingly being explored as a complementary approach to traditional cloud computing, offering decentralized processing capabilities that enhance responsiveness, particularly in latency-sensitive and edge-centric applications. This paradigm is especially relevant with the rapid growth of the Internet of Things (IoT) ecosystem, where vast amounts of data require real-time processing and low latency to support applications in smart cities, autonomous vehicles, healthcare, and industrial automation. This thesis explores the fog computing model extensively, providing an in-depth analysis of its architecture, primary components, applications, and the critical differences between fog and cloud computing. Central to fog computing is its multi-layered architecture, which includes the cloud, fog, and edge layers. These layers work collaboratively to address the limitations of centralized data centers, bringing data processing closer to its source to reduce latency, manage bandwidth, and enhance security and privacy.

Fog computing's architecture is structured to improve data processing and service delivery through a decentralized approach that operates at the network's edge. At the core of this architecture is the **fog node**, which interacts directly with end-user devices to provide essential services such as Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). By distributing computing tasks across local nodes, fog computing alleviates the dependence on distant cloud servers, reducing bandwidth requirements and enabling quicker response times. This characteristic of fog nodes, which are geographically dispersed and closer to data sources, enhances the quality of service (QoS) for latency-sensitive applications while supporting real-time analytics and improved mobility.

A comprehensive examination of fog computing's role in the Internet of Things (IoT) underscores its advantages in handling data generated by IoT devices, which has traditionally been managed by centralized cloud data centers. The thesis describes how, unlike cloud systems, fog computing enables the real-time processing of data generated by IoT sensors and devices by utilizing nearby fog nodes. This localization of processing tasks is essential for applications that cannot tolerate the latency associated with remote data centers, such as emergency response systems, real-time industrial monitoring, and autonomous vehicle networks. Furthermore, fog computing's distributed architecture supports enhanced scalability and can dynamically accommodate increased demands as IoT ecosystems continue to expand.

To highlight fog computing's versatility, this thesis presents a taxonomy of its **key features**, including context awareness, geographic distribution, and support for varied end-user devices. Fog computing nodes, capable of handling processing, storage, and communication tasks, are

typically positioned closer to the end-user, which mitigates the latency and bandwidth constraints associated with centralized cloud storage. The architectural flexibility of fog nodes facilitates a range of applications that rely on swift and reliable data access, such as smart city infrastructures and healthcare systems that require instantaneous data transmission to ensure efficient functioning. Fog computing nodes can handle tasks in a multi-layered setup, providing services at both the local and intermediary network levels, further optimizing resource allocation and management.

This research also investigates resource scheduling and management strategies in fog computing, emphasizing the importance of optimizing resource allocation to enhance performance. Fog computing environments, often resource-constrained due to limited processing and storage capacities, require effective scheduling mechanisms to ensure balanced load distribution and high system reliability. The thesis examines various resource management frameworks and scheduling algorithms, including heuristic-based approaches, optimization algorithms, and machine learning models, to address challenges associated with task distribution across fog nodes. By distributing tasks based on parameters such as latency, bandwidth availability, and energy efficiency, fog computing can maintain system responsiveness while minimizing energy consumption.

Energy efficiency is particularly crucial in fog environments where devices operate on limited power sources and are often deployed in locations with restricted access to continuous power. This thesis introduces an Energy-Efficient Resource Optimization (EERO) framework, which is specifically designed for scientific workflows within fog computing environments. The EERO model comprises three primary modules: initial processing, optimization, and parameter analysis. This multi-tiered approach facilitates the optimal use of available resources, significantly reducing execution time and energy consumption while supporting high-priority tasks.

The EERO framework applies advanced algorithms such as the Pareto distribution method for task prioritization and the PEFT ranking algorithm to dynamically allocate tasks across fog nodes. These techniques contribute to load balancing and reduce energy use by selectively processing tasks based on priority and resource availability. Through case studies and performance evaluations, this thesis demonstrates that the EERO model enhances fog computing's efficiency and scalability by providing an adaptable and robust resource management system. In contrast to cloud computing, fog computing supports **location awareness** and **localized data handling**, which enhances data privacy by processing sensitive information nearer to its source rather than transmitting it over the internet. This proximity also mitigates security risks associated with central cloud storage, where large-scale data

breaches are a significant concern. By managing data locally and securely, fog computing aligns with stringent data privacy regulations and supports compliance in industries such as healthcare and finance, where data protection is paramount.

To address dynamic resource management challenges, the thesis discusses several established and emerging scheduling techniques, including Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and Genetic Algorithms (GA). These bio-inspired algorithms provide efficient solutions for managing resource allocation in fog environments by mimicking natural processes to find near-optimal solutions for complex tasks. By integrating heuristic, optimization-based, and machine learning methods, these scheduling strategies ensure that fog computing can adapt to fluctuating workload demands and provide continuous service in diverse application settings.

The comparative analysis of fog and cloud computing reveals that while both paradigms offer scalable and flexible solutions for data processing, fog computing's proximity to end-users and low-latency capabilities make it more suitable for real-time applications. This thesis also reviews related paradigms such as mist and edge computing, positioning fog computing as an intermediary layer that bridges edge devices with centralized cloud services. In doing so, fog computing provides a hierarchical framework that supports scalability and efficient data management across a distributed network.

This work concludes with a discussion of future research directions, emphasizing the need for further development in areas such as task preemption, real-time resource migration, and advanced scheduling algorithms tailored to fog environments. As fog computing continues to evolve, addressing challenges related to resource scarcity, security, and interoperability will be crucial for its widespread adoption across industries. The findings presented in this thesis contribute to a deeper understanding of fog computing's potential to transform data processing frameworks and expand the capabilities of IoT, bringing a range of practical applications within closer reach.

CHAPTER 1

INTRODUCTION:

Fog computing allows compute and storage services to be provided closer to an organization's physical hardware, resulting in faster delivery times. This approach bypasses the broader Internet, which often depends on carrier speeds and network capacity, ensuring quicker and more efficient service delivery.

According to NIST Special Publication [31], fog computing is described as a physical, or virtual resource layer connecting traditional cloud computing or linked data centers and smart end devices typically found within organizations. The OpenFog Consortium [32] defines it as a architecture at the system level that divides up the processing, storage, control, and networking power closer to users. This highly virtualized platform connects traditional cloud data centers to end devices, providing networking, storage, and processing services. The localized nature of fog nodes reduces latency, enhances context awareness, and supports applications those are latency-sensitive by offering scalable, layered, ubiquitous, and federated network access.

Fog computing reduces latency and enhances context awareness by localizing fog nodes. It supports latency-sensitive applications through scalable, layered, federated, and pervasive network connectivity. Fog nodes provide similar services to cloud computing, such as (IaaS), (PaaS), and Software-as-a-Service (SaaS). The fog architecture involves significant communication, control, setup, measurement, and management functions via cooperative end-user clients or nearby edge devices. This paradigm widens cloud computing services to the network's edge, offering advantages over traditional cloud environments, which are often distant and dependent on larger Internet bandwidths. In contrast, fog services are nearer to end users, densely distributed geographically, and offer superior mobility support.

According to Gartner [1], the future of industrial IoT lies in edge-centric computing models, where research and system development focus on deploying processing capabilities near the source of data generation. As the Internet of Things (IoT) grows, so does the volume of data generated by these devices. Cloud computing data centres provide processing and storage services to these Internet of Things devices. Cloud computing allows for "pay-as-you-go" service delivery. Cloud computing data centres are distributed with a centralized organizational structure. Data processing and storage in data centres may take much longer than expected.

Sometimes, end devices find it difficult to retrieve the data in an emergency due to the centralized cloud storage. The Internet of Things (IoT) connects smartphones, smart cities, intelligent cars, and a host of other real-world things to the Internet, allowing data to flow between them with minimal human intervention. The Internet of Things sensors generate data relevant to specific applications and send it to the nearest sensor connections.

Cisco unveiled a new architecture in 2012 called fog computing to fulfil these IoT requirements. Consider fog computing as a network-edge development of cloud computing. Fog effectively completes tasks requiring low latency and minimal energy on vital computer nodes close to the network's centre. The fog computing concept was established to fulfil the needs of different Internet of Things (IoT), Internet of Everything (IoE), or Internet of Me (IoM) segments from start to finish, such as consumer, wearable, industrial, enterprise, vehicle, healthcare, building, and energy.

This chapter provides a high-level overview of this research project by describing the architecture, features, applications, advantages, and unsolved issues of fog computing andessential areas of interest. Programs for the scientific process have also been described. The need for resource scheduling in a fog environment has also been covered. Lastly, the order of the remaining chapters and contributions to the thesis have been provided.

1.1 Fog Computing Overview:

To expand the cloud, Cisco introduced fog computing, which provides services near end users. An ecosystem that uses fog computing allows many ubiquitous devices to connect without the assistance of third parties [2] [3]. The main objective of fog computing is to resolve problems that cloud computing encounters while handling Internet of Things data. The fog layer acts as a bridge between IoT devices and the cloud. It is a powerful technology that provides several answers to issues related to cloud computing. Decentralised fog offers networking, storage, and processing capabilities compared to centralised clouds [4]. The primary objective of fog computing is to resolve problems during cloud-based IoT data processing. This is a new paradigm that might be used with a variety of sensors, wearable technologies, smart gadgets, and cars. This paradigm states that jobs and computer tasks ought to be handled `in a dispersed fashion. Instead of building a single data centre, the network uses several devices. Starting with the end user and working your way up to the cloud reduces the bandwidth and latency of the network. Sensor-generated data is cleaned up by fog computing before being sent to the cloud. Several advantages come with this paradigm, such as improved IoT service analysis, monitoring, and execution speed [5].

1.1.1 Definition

Even though other scholars have proposed different interpretations, Cisco coined the term in 2012. From the perspective of Cisco, Fog brings cloud services closer to edge devices. “Fog computing is an architectural deployment of computing resources that employs distinct nodes for communication and data transfer amongst IoT devices instead of storing data from IoT devices in cloud data centres.” [6] According to F. Bonomi et al. [7], fog computing is a distributed, layered computing platform that provides end users with network, storage, and computation services. According to reports, fog computing works better for straightforward procedures and is comparable to IoT devices. [8]. The intelligent IoT data sensors and cloud data centres are connected by a fog layer, facilitating data execution and storage. Fog computing extends cloud services to rival the constraints of regular cloud computing. [9]. Smart cities, linked automobiles, connected homes, and intelligent healthcare are just a few technological components and applications that fog computing's hierarchical and distributed architecture can allow. [10].

The fog node is the central element of the fog computing environment that facilitates the operation of Internet of Things applications. The fog layer offers a few characteristics, including mobility, geological dispersion, and position awareness [11]. Fog computing, a type of decentralised computing technology, makes it possible to process and store data midway between the cloud infrastructure and its source. The continued growth of IoT devices primarily drives the fog computing paradigm. A growing variety of devices generate an increasing volume, diversity, and velocity of data [12].

The diagram presents a comprehensive taxonomy of fog computing, highlighting its multifaceted structure. At the core, fog computing encompasses various critical domains such as Security, Computing, Communication, and Management. Security is further divided into Encryption and Authentication, ensuring data integrity and protection. The Computing domain includes aspects like Storage and Services, which are essential for effective data handling and processing. Applications of fog computing span diverse fields, including Industrial IoT and Smart Cities, indicating its widespread utility. Communication focuses on Protocols and Interfaces, vital for seamless data exchange. The Computing Environment is distinguished by the presence of Edge Nodes and Fog Nodes, illustrating the distributed nature of fog computing. Management aspects cover Resource Management and Task Scheduling, crucial for efficient operation and maintenance. This taxonomy provides a

structured overview of fog computing's various elements, emphasizing its complexity and extensive applicability.

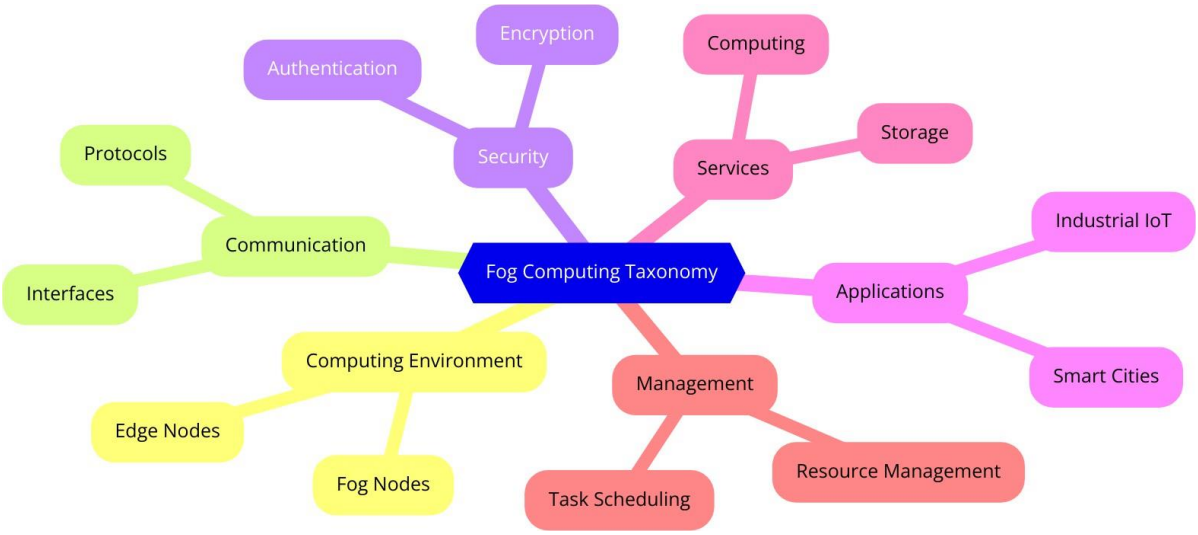


Fig 1.1: Taxonomy of Fog Computing

1.1.2 Fog Computing Development:

A developing technique called fog computing (FC) enhances current cloud computing (CC) capabilities to the network endpoints to provide lower latency through geographical distribution [13]. The devices in distributed computing employ a message-passing interface to support decentralised systems models in which numerous network devices perform all computational processes and simplify communication. Many new computer models have emerged in distributed computing. Utility computing comes before the notion of cloud computing. Cloud computing gained prominence in the early 2000s. Fog computing enables consumers to get information more quickly. The edge capacity of an application supports the computational capability of cloudlets to service various applications [14]. Tiny computer nodes called cloudlets, located close to customers' base stations, work with the fog and the cloud to provide a variety of applications. Fog computing applications are all developing in a way that makes high-performance computing (HPC) possible in networked systems.

When devices and users move from one point of access to another in these networked systems, all the data and processing associated with each user's computer typically relocate as well [15]. With data migration, users could find it simpler to access their data in an emergency. Delays in specific delicate settings, including transportation and healthcare systems, might lead to dangerous scenarios [16]. The fog computing paradigm provides all-

time-centric applications with rapid resource access—the better use of resources via management to get the highest output at the lowest possible cost. Effective resource management is crucial for various reasons, such as cost and response time. Applying fog computing in a real-time scenario is pretty challenging, though.

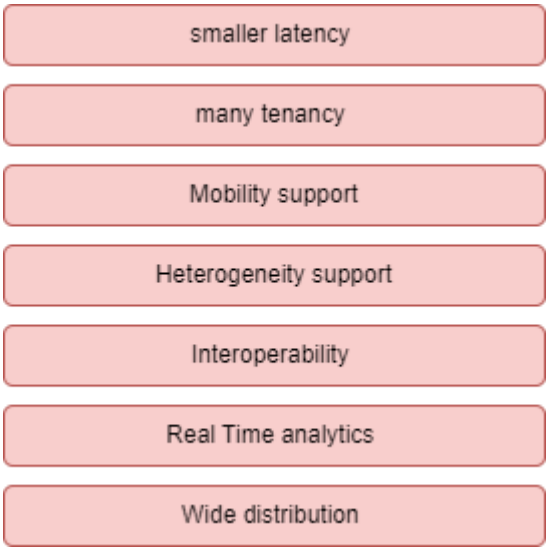


Fig .1.2 Characteristic of Fog Computing:

- Low latency- achieved through the proximity of fog nodes to on premise endpoint devices, enabling much faster response times and analysis.
- Varied end user support -Rich and varied end-user support due to Edge devices' proximity to compute nodes.
- Multiple tenancies in a regulated setting- Due to a highly virtualized distributed platform it increases direct contact between the Fog apps and mobile devices results in improved mobility assistance.
- Real-time interaction as opposed to batch processing, as is the case, for instance, with cloud-based apps.
- Contributes to the provision of high-quality streaming services. For time-consuming wireless sensing devices, wireless access networking makes more sense. Dispersed communication and analysis.

1.1.3 Dissimilarities of Cloud Computing and Fog Computing:

To complement cloud solutions and align with the evolving Internet of Things (IoT) vision, fog computing brings cloud capabilities to the network's edge. This distributed computing paradigm facilitates the operation of networking, storage, and processing services between

end devices and cloud data centres. In fog computing, application components typically run both in the cloud and on edge devices, such as smart gateways, routers, or devices specifically designed for fog computing.

Wireless networks have limitations, the Internet can be unstable, and the cloud requires substantial bandwidth. Fog computing significantly reduces the needed bandwidth by enabling data to be processed and transmitted within the local fog environment, minimizing reliance on the Internet. This allows most traffic, especially sensitive data, to stay off cloud networks, while critical data can still be transferred to the cloud. As a result, bandwidth is freed up for other cloud users.

Similar to cloud computing, fog computing provides storage, computation, and applications for end-users. However, fog computing is closer to end-users and has a broader geographical distribution. It emphasizes proximity to users, local resource pooling, and reduced latency, which improves quality of service (QoS) and enables edge analytics and stream mining. This leads to a better user experience. Fog computing extends the cloud concept to the network's edge, supporting applications and services that the cloud cannot accommodate due to technological and infrastructure limitations.

The volume of information in networking is continually increasing. To manage and distribute this data efficiently to end-users, services like cloud storage and cloud computing are utilized. However, for managing frequent security updates and mitigating bandwidth challenges, fog computing presents a more viable solution.

Table 1.1: Differences between Cloud Computing and Fog Computing

ASPECT	CLOUD COMPUTING	FOG COMPUTING
DEFINITION	A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources.	An architecture that uses edge devices to carry out a substantial amount of computation, storage, and communication locally.
ARCHITECTURE	Centralized architecture with data and processing in a central cloud server.	Decentralized architecture with processing distributed across edge devices and local nodes.
LATENCY	Higher latency due to data traveling to and from a central cloud.	Lower latency as data processing is closer to the data source.
PROCESSING LOCATION	Data processing occurs in centralized data centers.	Data processing occurs at the edge of the network, closer to the data source.
SCALABILITY	Highly scalable with virtually unlimited resources.	Scalable but within the limits of local resources and network capabilities.
DATA MANAGEMENT	Centralized data management with large-scale data storage and processing.	Decentralized data management with data processed and stored closer to where it is generated.
SECURITY	Security managed by central cloud providers, with strong but centralized security protocols.	Enhanced security due to data being processed locally, reducing the risk of centralized attacks.

IDEAL USE CASES	Suitable for tasks requiring heavy computation and large-scale data storage, like big data analytics and machine learning.	Suitable for real-time applications requiring low latency, like IoT, smart grids, and autonomous vehicles.
EXAMPLE TECHNOLOGIES	AWS, Google Cloud, Microsoft Azure	Cisco Fog Computing Solutions, Nebbiolo Technologies, Dell Edge Gateway

1.1.4 Fog Architecture:

A basic fog computing architecture consists of three levels. The uppermost layer is the Internet of Things, which houses intelligent gadgets. The second layer, fog computing, comprises fog nodes with constrained processing and storage power.

The architecture describes the interaction between edge devices and the cloud, forming a unified system that bridges these two components. It typically follows a three-layered structure, detailed as follows:

Layer 1: This is the foundational layer, encompassing all Internet of Things (IoT) devices. These devices are responsible for gathering and transmitting raw data to the next layer.

Layer 2: Positioned in the middle, this layer features networking devices such as routers and switches. It handles the preliminary processing of data and offers temporary storage. These devices are connected to the cloud and continuously send data at regular intervals.

Layer 3: This is the topmost layer, comprising servers and data centers. It is equipped to store substantial volumes of data and has the capability to process it efficiently.

In addition to this, the initial layer includes both physical and virtual nodes. Various sensors are employed to track environmental conditions, transmitting the collected data to upper layers through gateways for further processing. The **monitoring layer** manages task requests and oversees energy consumption of the core physical devices. The **pre-processing layer** handles data management tasks, such as filtering and cleaning. The **temporary storage layer** provides short-term data retention. The **security layer** is dedicated to encrypting and decrypting data, ensuring integrity and protection against tampering. Finally, the **transport layer** forwards the processed data to the cloud, enabling the cloud to extract valuable insights from it[17].

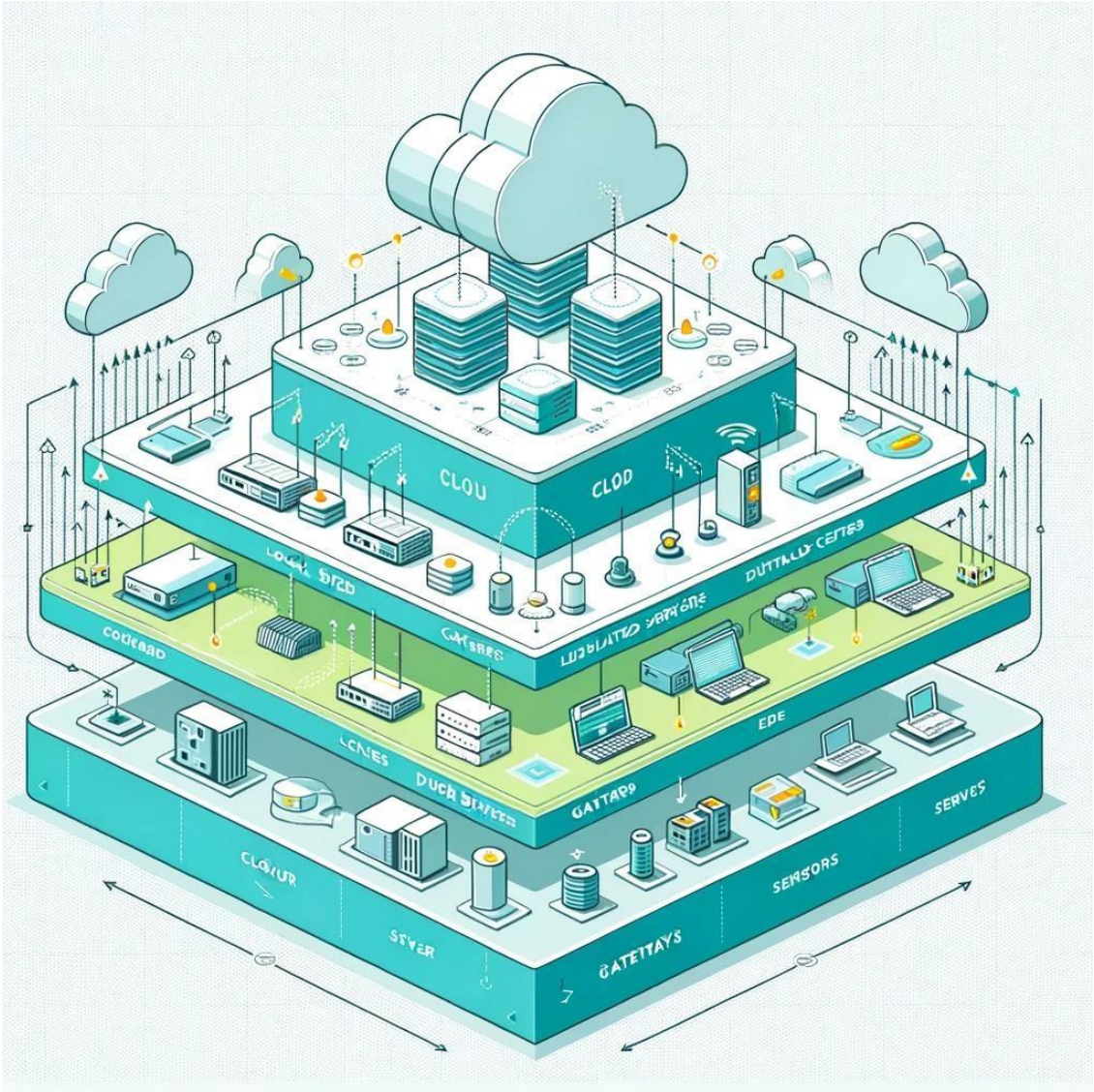


Fig 1.3 Layered architecture of Fog computing

Layered Architecture of Fog Computing

1. Cloud Layer (Top Layer)

- **Description:** This layer represents large, centralized servers or data centers.
- **Appearance:** It features cloud icons and infrastructure symbols, using light blue and white tones.
- **Function:** The Cloud layer handles extensive data processing, storage, and complex computations. It serves as the central control point and provides overarching services to the Fog and Edge layers.
- **Connections:** Arrows point downward to the Fog layer, indicating the transmission of data and commands from the Cloud to the Fog nodes.

2. Fog Layer (Middle Layer)

- **Description:** This layer consists of distributed and decentralized nodes such as local servers, gateways, and edge devices.
- **Appearance:** Depicted with small server icons, router symbols, and intermediary devices in shades of green and light grey.
- **Function:** The Fog layer acts as an intermediary, processing data closer to the source (Edge devices) to reduce latency and bandwidth usage. It provides local processing, storage, and control functions.
- **Connections:** Arrows point upward to the Cloud layer and downward to the Edge layer, indicating bidirectional data flow between the Cloud and Fog, and from the Fog to the Edge. Some nodes in this layer are connected laterally to show inter-node communication within the Fog layer.

3. Edge Layer (Bottom Layer)

- **Description:** This layer includes end-user devices such as smartphones, laptops, sensors, and IoT devices.
- **Appearance:** Illustrated with icons representing various personal and industrial devices in orange and yellow tones.
- **Function:** The Edge layer is the point of data generation and initial processing. Devices in this layer collect and perform preliminary processing on data before sending it to the Fog layer for further processing.
- **Connections:** Arrows point upward to the Fog layer, indicating the transmission of data from Edge devices to Fog nodes.

1.1.5 Related computing models:

Fog computing is part of a broader landscape of distributed computing models, each with its unique characteristics and use cases. Below, I outline some related computing models and their key differences:

1. Cloud Computing
2. Edge Computing
3. Fog Computing
4. Mist Computing

Cloud Computing

Definition: Cloud computing involves delivering computing services (such as servers, storage, databases, networking, software) over the internet ("the cloud").

Characteristics:

- Centralized resources in large data centers.
- High scalability and flexibility.
- Pay-as-you-go pricing models.
- Services accessed via the internet.

Use Cases:

- Large-scale data storage and processing.
- Web hosting and application development.
- Big data analytics and machine learning.

Key Differences:

- Cloud computing is highly centralized, whereas fog and edge computing distribute resources closer to the data source.
- Cloud computing may experience higher latency due to the distance between users and data canters.

Edge Computing

Definition: Edge computing refers to processing data at or near the source of data generation, minimizing latency and bandwidth usage.

Characteristics:

- Decentralized processing at the network edge.
- Low latency and real-time processing.
- Reduced bandwidth consumption.
- Enhances privacy and security by keeping data local.

Use Cases:

- IoT devices and smart sensors.
- Autonomous vehicles and industrial automation.
- Real-time analytics and augmented reality.

Key Differences:

- Edge computing focuses on processing at the very edge of the network, such as directly on devices or local gateways.
- Fog computing extends edge computing by adding an additional layer of intermediate processing between the edge and the cloud.

Fog Computing

Definition: Fog computing is a decentralized computing infrastructure where data, compute, storage, and applications are distributed in the most logical, efficient place between the data source and the cloud.

Characteristics:

- Intermediate layer between edge and cloud.
- Processes data closer to the source than cloud computing but may aggregate data from multiple edge devices.
- Reduces latency and bandwidth usage.
- Enhances security by local data processing.

Use Cases:

- Smart cities and connected vehicles.
- Healthcare monitoring and management.
- Industrial IoT and real-time analytics.

Key Differences:

- Fog computing provides a hierarchical layer between edge and cloud, offering distributed computing closer to the source while still enabling cloud integration.
- More suitable for applications requiring low latency and high reliability but benefiting from cloud capabilities.

Mist Computing

Definition: Mist computing is an even more localized form of computing, often considered the "micro" level of fog computing, where data processing occurs directly on microcontrollers and small devices.

Characteristics:

- Extremely localized processing on micro-level devices.
- Ultra-low latency.
- Minimal reliance on network connectivity.
- Suitable for simple, real-time processing tasks.

Use Cases:

- Wearable devices and smart sensors.
- Simple IoT applications requiring immediate responses.
- Local data filtering before sending to fog or cloud.

Key Differences:

- Mist computing operates on a much smaller scale than fog and edge computing, focusing on the immediate vicinity of the data source.
- Often used for preliminary data processing before sending data to fog or edge layers.

Comparative Table 1.2

Feature	Cloud Computing	Edge Computing	Fog Computing	Mist Computing
Centralization	Centralized	Decentralized	Intermediate (between cloud and edge)	Highly localized
Latency	Higher	Very low	Low	Ultra-low
Processing Location	Data canters	At or near data source	Intermediate nodes between cloud and edge	On micro-devices
Scalability	High	Moderate	High	Low
Use Cases	Large-scale applications	Real-time applications	Real-time and aggregated data applications	Immediate response applications
Examples	Web hosting, big data	Autonomous vehicles, AR/VR	Smart cities, healthcare	Wearables, smart sensors

1.1.6 Resource Allocation in Fog Computing: Resource allocation in fog computing involves the efficient distribution of computing, storage, and network resources to various applications and services running on fog nodes. This is critical for ensuring low latency, high availability, and optimal performance of applications. Below is an overview of the key aspects and strategies involved in resource allocation in fog computing?

Key Aspects of Resource Allocation

1. **Resource Types**:
    - **Computing Resources**: CPU, GPU, and memory resources required for processing tasks.
    - **Storage Resources**: Local storage for data caching, databases, and file systems
    - **Network Resources**: Bandwidth and network interfaces for communication between devices and nodes

2. **Allocation Strategies**:
    - **Static Allocation**: Resources are allocated in advance based on predefined rules and configurations. This approach is simpler but less flexible.
    - **Dynamic Allocation**: Resources are allocated on-demand based on real-time requirements and conditions. This approach is more complex but offers better efficiency and adaptability.
3. **Optimization Goals**:
    - **Minimize Latency**: Ensuring that data processing and communication occur with minimal delay.
    - **Maximize Throughput**: Enhancing the amount of data processed in a given time period.
    - **Energy Efficiency**: Reducing power consumption while maintaining performance.
    - **Load Balancing**: Distributing workloads evenly across available resources to avoid bottlenecks and overloading.

4. **Challenges**:
    - **Heterogeneity**: Diverse devices and resources with varying capabilities and performance.
    - **Mobility**: Devices and users may move, requiring dynamic reallocation of resources.
    - **Scalability**: Managing a large number of devices and applications efficiently.
    - **Security**: Ensuring data privacy and security during allocation and processing.

Resource Allocation Strategies

1. **Heuristic-Based Approaches**:
    - Use rule-based methods and heuristics to allocate resources. Examples include round-robin, first-fit, and best-fit algorithms.
    - Pros: Simplicity and ease of implementation.
    - Cons: May not provide optimal solutions in complex scenarios.
2. **Optimization-Based Approaches**:
    - Use mathematical models and optimization techniques (e.g., linear programming, integer programming) to find optimal resource allocation.
    - Pros: Can provide near-optimal solutions.
    - Cons: Computationally intensive and may not scale well.
3. **Machine Learning-Based Approaches**:
    - Use machine learning models to predict resource demands and allocate resources accordingly.
    - Pros: Can adapt to changing conditions and improve over time.
    - Cons: Requires training data and computational resources for model training and inference.
4. **Game Theory-Based Approaches**:
    - Use game theory to model the interaction between different entities (e.g., devices, fog nodes) and allocate resources based on equilibrium strategies.
    - Pros: Suitable for decentralized and distributed environments.

1.1.7 Resource Optimization in Fog Computing

Resource optimization in fog computing is all about making sure that we use our computing resources—like processing power, storage, and network bandwidth—in the most efficient way possible. This is key for ensuring that the applications and services running on fog nodes

perform well and meet user expectations. Below, we'll explore key strategies and techniques for optimizing resources in fog computing, along with some references to important studies and articles on the topic.

Key Strategies for Resource Optimization

- Load Balancing
- Energy Efficiency
- Quality of Service (QoS)
- Resource Scheduling
- Data Placement
- Latency Reduction

1. **Load Balancing**

Load balancing is a technique to distribute tasks evenly across multiple fog nodes so that no single node gets overloaded [33]. Load balancing in fog computing is essential for efficiently distributing tasks and workloads across multiple fog nodes. This process ensures that no single node is overwhelmed, which helps maintain optimal performance and reliability. By evenly spreading out the computational demands, load balancing minimizes latency and improves response times, making it a critical component for delivering smooth and seamless services to end-users. Additionally, it enhances resource utilization and energy efficiency across the network, contributing to a more robust and scalable fog computing environment. It prevents any single node from becoming a bottleneck, ensures high availability and reliability, and enhances overall system performance.

2. **Energy Efficiency**

Reducing the energy consumption of fog nodes while still maintaining good performance. Energy efficiency in fog computing is about optimizing the use of resources to minimize power consumption while maintaining high performance. By processing data closer to where it's generated, fog computing reduces the need for long-distance data transmission, which can be energy-intensive. This local processing not only speeds up response times but also cuts down on the energy used by central data canters. Efficient resource management and dynamic task allocation further enhance energy savings, making fog computing a greener, more sustainable solution for modern computing needs.

Techniques

Dynamic Voltage and Frequency Scaling (DVFS): Adjusts the power and speed of processors based on the current workload.[34]

Task Consolidation: Groups tasks to run on fewer nodes, allowing some nodes to enter low-power states.

Energy-Aware Scheduling: Allocates tasks to nodes based on their energy efficiency.

Why It Matters:

- Extends the lifespan of fog nodes.
- Lowers operational costs.
- Supports environmental sustainability.

3. **Quality of Service (QoS)**

Ensuring that fog computing services meet specified performance metrics like latency, throughput, and availability. In fog computing, Quality of Service (QoS) is essential to the seamless and effective operation of applications. For real-time applications, it entails managing network resources to ensure fast and dependable data transmission. Fog computing can maintain high performance and minimising latency by setting priorities for jobs and optimising resource allocation. By doing this, users are guaranteed stable and constant service regardless of the fluctuations in the network. To put it simply, quality of service (QoS) in fog computing refers to providing optimal user experience while maintaining resource efficiency, speed, and dependability [34].

Resource Scheduling:

Resource scheduling in fog computing is all about efficiently managing and allocating tasks to various fog nodes to maximize performance and minimize delays. It involves determining the best way to distribute computational loads based on the availability and capacity of nearby nodes. Effective resource scheduling ensures that tasks are handled promptly and that resources are not underutilized or overburdened. This not only enhances the overall system efficiency but also improves the user experience by providing quicker response times and maintaining smooth operation across the network. Efficiently scheduling tasks and resources to optimize performance and utilization[35].

Static Scheduling:

Uses pre-determined schedules based on known workloads.

Dynamic Scheduling: Adjusts schedules in real-time based on current system state and workload demands.

Predictive Scheduling: Uses historical data and machine learning to predict and schedule future workloads. It Improves resource utilization, reduces waiting times for tasks and enhances system responsiveness.

Data Placement Strategically placing data close to where it's needed to minimize latency and bandwidth usage.

Techniques :

Data Caching: Stores frequently accessed data on local nodes.

Data Replication: Creates multiple copies of data across different nodes for redundancy and faster access.

Proximity-Aware Placement: Places data based on the geographic location of data sources and users.

It reduces data access latency, optimizes bandwidth usage and enhances data availability and reliability.[36]

Latency Reduction

Minimizing the delay between data generation and processing to support real-time applications. Latency reduction in resource optimization in fog computing focuses on minimizing the delay in data processing and transmission. By processing data closer to the source, fog computing significantly cuts down the time it takes for data to travel to and from centralized data centers. This local processing means quicker response times and more efficient handling of time-sensitive tasks. Optimizing resources effectively across fog nodes further helps in reducing latency, ensuring that applications run smoothly and users experience minimal delays, enhancing overall system performance.

Techniques:

Edge Processing: Performs data processing close to data sources.

Fog Node Hierarchies: Creates multiple layers of fog nodes to process data progressively closer to the data source.

Latency-Aware Task Allocation: Allocates tasks to nodes based on their proximity to the data source and processing capabilities.

It Supports real-time applications like autonomous vehicles and IoT, Improves user experience with faster response times.

Here is a diagram that shows how these strategies fit together in a tree-like structure:

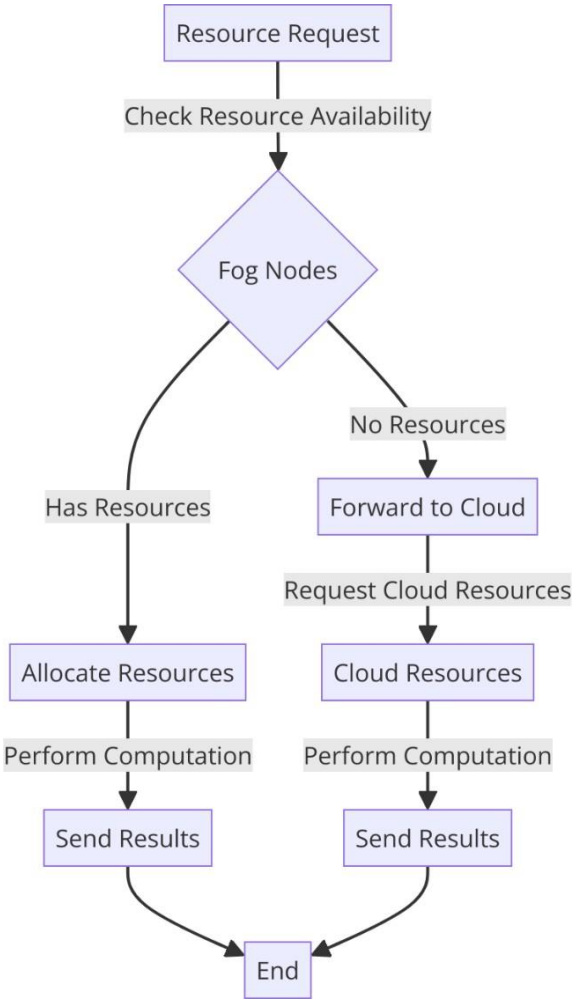


Fig 1.4 Resource Optimization in Fog Computing

Key Areas focused by Fog computing:

1.1.8 Advantage of Resource Management in Fog computing:

❖ Optimized **Resource Utilization:**

- **Benefit:** Effective resource management ensures that computing, storage, and network resources are used efficiently, reducing waste and maximizing performance.

- **Explanation:** By dynamically allocating resources based on current demands, fog computing can handle varying workloads more effectively. This ensures that resources are not left idle and are instead utilized to their full potential, leading to better overall system efficiency.

❖ Improved **Quality of Service (QoS):**

- **Benefit:** Enhanced QoS through better resource management ensures that applications meet their performance requirements, such as low latency and high availability.
- **Explanation:** Resource management techniques prioritize critical tasks and allocate resources accordingly, ensuring that important applications maintain their performance standards even during peak usage times.

❖ Cost **Savings:**

- **Benefit:** Efficient resource management reduces operational costs by optimizing the use of available resources and minimizing the need for additional infrastructure.
- **Explanation:** By making better use of existing resources and reducing unnecessary data transmission to the cloud, fog computing can lower expenses related to bandwidth, storage, and processing power.

❖ Scalability**:**

- **Benefit:** Effective resource management allows fog computing systems to scale easily to accommodate growing data and computational demands.
- **Explanation:** As the number of connected devices increases, resource management strategies enable the system to adapt and scale without significant performance degradation, ensuring smooth operation as the network expands.

❖ Energy **Efficiency:**

- **Benefit:** Resource management in fog computing can lead to more energy-efficient operations, reducing the overall energy consumption of the network.
- **Explanation:** By optimizing the allocation of resources and minimizing unnecessary data processing and transmission, fog computing systems can operate more sustainably, conserving energy and reducing their environmental impact.

❖ Enhanced **Security and Privacy:**

- **Benefit:** Localized resource management enhances security and privacy by processing sensitive data closer to the source and minimizing exposure to external threats.
- **Explanation:** By managing resources at the edge, fog computing reduces the need to transmit sensitive data to centralized cloud servers, thus lowering the risk of data breaches and ensuring better compliance with privacy regulations.

❖ Reduced **Latency:**

- **Benefit:** Proper resource management significantly reduces latency by processing data closer to where it is generated.
- **Explanation:** Fog computing minimizes the distance data must travel, enabling faster data processing and response times, which is critical for real-time applications such as autonomous vehicles and smart grids.

❖ Increased **Reliability:**

- **Benefit:** Resource management enhances system reliability by ensuring that resources are available where and when they are needed.
- **Explanation:** By distributing resources across multiple nodes and dynamically adjusting to changes in demand, fog computing can maintain high availability and continue operating effectively even in the face of individual node failures.

Chapter 2

Related Work

The fog computing paradigm integrates conventional technology to support Internet of Things environments that create large amounts of data. It has become a remarkable technology that enables creative applications and excellent performance in demanding situations. The appropriate use of dynamically and spatially dispersed resources across the system is necessary for deploying fog devices at the fog layer. The Internet of Things is becoming more and more critical every day, leading to a rise in the demand for massive power handling, rapid internet systems, and information storage to handle data streams. Fog Computing has now fulfilled these prerequisites. One of the primary responsibilities of fog computing is resource management and effective utilization. The resource management system provides resource scheduling and provisioning to assist in resource management choices.

Managing resources and making effective use of fog computing is essential. The resource management system provides resource scheduling and provisioning to assist in making resource management choices.

Various research has focused on the issue of process scheduling in diverse computing systems. Because of the its nature that is NP-hard, it is necessary to employ heuristic approaches to estimate optimal solutions.

The authors of [22] highlight a problem in scheduling workflows with multiple objectives in Hybrid-cloud systems. The optimization goals are considered time efficiency, cost-effectiveness, and reliability from the user's perspective. Unlike traditional multi-parameter scheduling issues in the cloud, the mentioned technique allows clients to create a different approach to enhance reliability. This study presents a reliability-aware multi-objective mimetic algorithm (RA MOMA) incorporating a unique method and a diversification technique to address the Hybrid cloud issues. The diversification strategy employs many problem-related specific genetic algorithms to produce offspring with diverse features.

Regarding the technique, four problem-specific neighbourhood operators are developed based on the resource utilization rate and critical path. The purpose is to enhance the quality of the archive collection. A comprehensive statistical experiment is conducted to assess the

effectiveness of RA-MOMA. RA-MOMA outperforms similar methods in solving the MOWSP-MCS, as evidenced by comparisons with these algorithms.

The Heterogeneous Earliest Finish Time (HEFT) algorithm is widely used for scheduling tasks [27]. HEFT consists of two distinct phases: job prioritizing and selection of processes. The first phase involves assigning their priorities to tasks based on their ascending ranks. In contrast, the second phase selects an appropriate processor for task execution, taking into account the least time taken by a task to complete. Next renowned algorithm in this classification is Predict Earliest Finish Time (PEFT) [6]. PEFT utilizes the OCT which is Optimistic Cost Table to prioritize tasks and select the optimal processor for executing task during the scheduling phase. Both HEFT and PEFT are a kind of optimization which focuses on single objective that minimizes makespan. In contrast, EM-MOO focuses on multi-objective optimization strategy that concentrates on energy usage and makespan.

The paper [16] presents the Minimal Optimistic Processing Time (MOPT) approach, which aims to minimize the makespan. This technique alters the prioritizing phase by calculating tasks Optimistic Processing Times (OPT) on all executing nodes. The tasks are then ranked according to their average OPT values. The node selection phase enhances the entry task duplication feature by permitting duplication only if it lowers the completion time of subsequent jobs. Once again, it is worth noting that MOPT is a single-objective optimization strategy, in contrast to the approach we suggest in this paper. In [7], a hybrid meta-heuristic strategy is proposed to minimize the makespan in a multi-processor cloud environment. This approach combines the Genetic Algorithm (GA) with Ant Colony Optimization (ACO). The lowest level (b-level) of a task is utilized for assigning priorities. The b-level represents the most significant amount of time it takes for a job to traverse all levels of the graph. Subsequently, the Ant Colony Optimization (ACO) algorithm is utilized to determine an appropriate route, which is subsequently enhanced by applying the Genetic Algorithm (GA).

The paper given in [27] is one of the rare papers that examines job scheduling in fog computing as a Directed Acyclic Graph (DAG) scheduling problem. The paper presents the Cost-makespan-aware Scheduling (CMaS) method, which aims to meet the user's QoS criteria for optimizing both makespan and cost. It also introduces a utility function that helps identify the balance between these two objectives. The schedule is enhanced through the work reassignment step. The Task Scheduling in Fog Computing (TSFC) technique relies on the classification mining algorithm [19]. The association rules derived from the I-Apriori algorithm are integrated with the task completion durations, disregarding the bandwidth between machines. The scheduling of tasks in fog computing-supported software-defined

embedded systems (FC-SDES) [38] aims to minimize the makespan. The algorithm suggests a 3-phase approach that combines work scheduling, resource management, and I/O request balancing while minimizing complexity.

Workflow scheduling algorithms must consider resource attributes and dependency constraints in order to maximize effective resource utilization or make the best use of the resources at hand while limiting the application's overall completion time or makespan. It is a well-known NP-complete problem [24] that must be optimized using approximations in almost polynomial time [25]. In addition to makespan, energy consumption is another vital element in a fog-cloud environment. Over the past decade, the energy usage of cloud data centers has significantly risen, leading to a substantial increase in economic and operational costs and environmental consequences. Additionally, the restricted availability of resources in fog nodes presents a significant energy barrier. These nodes typically operate on batteries or have access to limited (renewable) energy sources. They are often deployed in places with limited and inconsistent energy supplies [?]. Consequently, there has been considerable focus on green cloud computing in academic and industrial circles. A key concern is decreasing energy usage in the growing fog-cloud infrastructure .

This chapter describes resource management and talks about fog computing systems. And the use of resources. It also covers essential background information to help with a better understanding of resource scheduling in fog computing. An examination and comparison of the To better understand resource scheduling and resource utilization in the fog environment, research has been done on the existing resource scheduling strategies.

2.1 Resource Management

Fog computing minimizes the quantity of data sent to the cloud. for processing, analysis, and storage, improving speed and efficiency. Quality of service (QoS) in fog computing describes the general performance of a service, especially as perceived by network users.

An evaluation of several network service components, including throughput, latency, resource availability is typical when assessing the quality of service.

It might take a long time to work because it gets stuck in long lines or takes a less direct route to avoid delays. Because of the need for applications that can't wait to process and move data in real-time, cloud computing tries to bring cloud services and tools to the network's edge [13].

Based on Quality of Service (quality of service) needs, this new way of doing things i.e fog computing which moves computing around between cloud places and network parts. Even

though cloud technology has been studied a lot, IoT services with specific needs can only be used if cloud resources are physically far away from end users. Many different experts have come up with various quality of service factors. In an IoT based on the cloud, the delay from endpoints talking to each other is further from the quality of service that is wanted for real-time services. When something goes wrong, healthcare services may need to move right away.

In the same way, self-driving cars need to be able to notice when their surroundings change quickly. Two of the most sensitive IoT uses are real-time tracking in factories and real-time guidance in traffic control systems. The quality-of-service monitor figures out how long the network is taking compared to other system nodes and keeps an eye on the worker node's quality of service features, like how available it is and how many resources it uses.

Moreover, the assessment of the impact of regulatory actions on the quality of service is crucial for the effective implementation of real-time services, which demands a very small latency in the allocation of dispersed resources. QoS-aware service allocation may be greatly impacted by control decision latency, which is mostly determined by the control topology.

2.2 Use of Resources:

Resource utilisation is referred to by a number of names, including scheduling and resource provisioning. Terms related to resource scheduling and utilisation are employed in the framework for carrying out the suggested job. The practice of ensuring that resources are as useful as possible in order to effectively fulfil user requirements is known as resource utilisation. Achieving optimal resource allocation and distributing all available resources among users can lead to high resource utilisation. Any system's cost and performance are directly impacted by how its resources are used. Under-provisioning resources can result in a system with worse performance. In contrast, over-provisioning resources can lead to low utilisation of all allocated resources and raise the cost of the system [14].

Many terms are linked to resource usage, such as scheduling and provisioning. There are terms for using resources and planning when to use them for the framework of the planned work. Resource usage is ensuring that resources are used as efficiently as possible to meet user needs. To achieve high resource utilization, it is necessary to ensure that all available resources are shared among all people. How well resources are used directly affects how much a system costs and how well it works. If you give too many resources to a system, they might not be used at all, and the cost of the system might go up. On the other hand, if you give more resources, the system might work better.

2.2.1 Fog computing resource utilisation:

This section reviews the resource utilisation strategies that are currently in use in fog computing. A fog computing paradigm is proposed to address all the problems associated with resource distribution for Internet of Things related applications. The scheduler for Bag-of-Task applications, or BaTS, was proposed by researchers in [70]. BaTS monitors the progress of operations and dynamically reconfigures the equipment to meet the demands. They conducted a number of experiments using a price-to-performance ratio. Every test was run on two different clouds, one using the BaTS algorithm and the other using the RR (Round-Robin) method. A fog system based on clouds was depicted in [33]. A simulation was established in this study using requirements for discrete events. To lessen the usage of the cloud, no specific load-balancing method is applied.

To give an energy-efficient solution, researchers in [15] represented the Energy-Efficient Task Scheduling (MEET) method for identical nodes. Their selection process and offloading time slot allotment resulted in a decrease in overall energy use. The author presented a greedy knapsack scheduling (GKS) method in [16] for resource allocation in a fog enabled network. Their study's outcome was reproduced in two case studies. Their suggested approach produced better results than the FCFS, and delay-priority algorithms. Applications about containers were given access to a network-oriented scheduling approach [17]. They reduced network latency by 70% using a fog computing architecture built on Kubernetes. The author of[18] suggested a hybrid approach for service orchestration in fog networks. South Bound and North Bound were the two new stages—a choreographic method allowed for automated and quick decision-making at the South-Bound level. Conversely, North-Bound employs centralised orchestration at both the cloud and fog layers.

In fog computing, Zeng et al. [19] introduced a scheduling technique in addition to picture placement. Fog nodes and embedded clients may complete all computational processes via storage servers. Clients and fog nodes can access the job image stored on the storage server. It is possible to reduce the completion time by planning each job. Ni et al. [20] introduced a dynamic resource allocation system based on the time required to complete each job and to enhance resource utilization. To increase the trustworthiness of fog nodes and to improve user quality of service, a method known as Priced Timed Petri Netts (PTPNs) was employed. A resource allocation strategy for optimizing energy usage was described by Pooranian et al. []. The algorithm was heuristically based. The resource allocation mechanism the author devised is called the "bin packing penalty" and is represented by fog servers. Every virtual

machine was used by time and frequency constraints. An additional strategy, the "penalty and reward policy," is employed to optimize energy usage.

Sun et al.[95] presented a two-level resource scheduling approach. According to these authors, distributing resources among several fog clusters produced a brief delay. The theory of enhanced non-dominated generic algorithm-II was scheduled by fog nodes assigned to different clusters. Resource scheduling amongst fog nodes was put into practice for multi-objective optimization.

To raise awareness about vehicular networks, [] suggested an integrated structure. To test the flexibility, they have been turning to the OMNeT++ framework. A blockchain-based consensus sensing (CS) application was created for this study to reconcile local data.

A unique bio-inspired hybrid algorithm was presented by Rafique et al. [78] for effective resource management in fog computing. The task above allocated and managed the resources according to the volume of incoming requests. The primary goal of this effort was to schedule the jobs efficiently to minimize the familiar or averge response time and maximize resource use. Task scheduling was handled by the scheduler installed between the fog nodes and the devices. The inefficient scheduling of given tasks was resolved by integrating a technique, called Modified Particle Swarm Optimisation (MPSO) and Modified Cat Swarm Optimisation (MCSO). This method was verified, and the outcomes demonstrated that it is more accurate at scheduling the jobs.

The studies listed above have all suggested scheduling algorithms; they have yet to address how user requests change dynamically in the ever-changing cloud-fog environment. Based on an analysis of recent research, it has been determined that the field of cloud computing is the primary focus for studying dynamic resource management. A job scheduling algorithm was developed using the ant colony system and a mixture of laxity in a cloud-fog environment [109]. The laxity metric was employed to ascertain job priority, while the ant colony algorithm was utilised for task scheduling. The intermediary cloud fog broker, situated between the cloud and fog layers, was tasked with assigning duties according to their criteria. At first, the requests from the IoT devices are broken down into tasks and the computing estimation of each job is performed to establish the nature of the work. Subsequently, the duty was assigned to either fog or cloud. The efficacy of this strategy in work scheduling was confirmed by rigorous testing and analysis of the output. The task allocation was executed efficiently using the given technique; nevertheless, it fails to account for the dynamic nature of the tasks, resulting in a bottleneck issue when the task count is raised.

The multi-level feedback queuing was suggested as a method for job offloading in the fog computing framework, considering both deadline and priority [9]. At first, the jobs were categorized into three groups according to their deadlines: high-priority, middle, and low-priority activities. The virtual queue idea was utilized to organize and prioritize work. If the tasks having less priority are not executed within a specific timeframe, their priority will be increased by one. The procedure was performed, demonstrating its efficacy in job categorization and scheduling. The deadline above and the priority-conscious task scheduling approach show task categorization and scheduling proficiency. However, it failed to consider the energy efficiency of the process and did not consider the selection of the fog node based on energy and resource availability.

In his discussion of over- and under-provisioning, Agarwal [29] suggested an architectural solution to address the issue in the fog environment. The most efficient way to employ processing time and allocate resources to programs is through scheduling. The primary responsibility of scheduling is to implement a set of applicable processes and determine which process to execute in the next iteration

A layered fog node architecture designed by Aazam et al. [30](Aazam and Huh, 2014) enables the processing of local service requests. Intelligent gateways and a data encryption layer have been installed in fog nodes. They created an intelligent network and smart gateway using a tiered architecture. They used a range of physical nodes, virtual nodes, wireless sensor networks, and virtual sensor networks to administer the system.

The combination of A3C learning and residual recurrent neural networks was used in edge cloud computing environments to execute dynamic scheduling [98]. The IoT devices' duties were dynamically planned by implementing a resource management system. The Resource Management System (RMS) determines the scheduling of tasks by considering factors such as CPU use, memory requirements, bandwidth availability, projected completion time, and deadline. The Resource Management System (RMS) comprised a Deep Reinforcement Learning (DRL) model for forecasting the subsequent scheduling determination. At the same time, the Constraint Satisfaction Module (CSM) would verify the limitations and offer feasible migration and scheduling determinations. The loss values were utilised to modify the parameters, whereas the R2N2 was employed to adjust the model parameters of the DRL model. The forecasting of the subsequent scheduling determination is quite effective. It decreases the average reaction time of the process, but the loss function of the forthcoming scheduling task diminishes the efficiency of the process.

This part looks at how resources can be used in fog computing. Several fog computing method has been proposed to solve all the problems of allocating resources for Internet of Things (IoT) apps.

First-Come-First-Served (FCFS),This technique is simple to implement and easy to understand but can lead to long waiting times for short tasks if a long task arrives first [43].

In Round robin, It is fair to all tasks and simple to implement; however, context switching overhead can be high [44].

Priority Scheduling-Executes high-priority tasks first, which is beneficial for critical applications, but lower priority tasks may suffer from starvation [45].

Shortest Job Next (SJN) -Can minimize the average waiting time but requires knowledge of execution time in advance, which may not always be possible [46].

Ant Colony Optimization (ACO)-This bio-inspired algorithm can find near-optimal solutions and is adaptable to dynamic changes but is computationally intensive and may require significant time to converge [47].

Particle Swarm Optimization (PSO)- Efficiently explores the search space and is good for handling dynamic environments; however, it requires fine-tuning of parameters, and convergence may not be guaranteed [48].

Dynamic Least Load First (DLLF)- Balances the load effectively and reduces the chances of any single node becoming a bottleneck but may not always result in the shortest total execution time for all tasks [49]

Genetic Algorithms (GA)-Capable of finding high-quality solutions for complex problems but can be computationally expensive and may require significant time to reach an optimal solution [50].

Heuristic-based Scheduling -Fast and effective for specific types of tasks or environments but may not always find the best possible solution, and performance is highly dependent on the quality of the heuristics used[51].

Researchers in [54] came up with the idea of BaTS, which stands for "budget-constrained scheduler for Bag-of-Task applications." BaTS keeps an eye on how operations are changing and changes the configuration of the machines on the fly based on what is needed. They did many tests with a price-performance ratio. Two different clouds were used for each test. In one (Round-Robin) algorithm was used, and for the other one used BaTS. In [33], a cloud-based fog device was shown. A simulation was set up based on discrete event requirements for this study. The author's load-balancing technique to cut down on cloud use needs to be clarified.

A resource allocation strategy for optimising energy consumption was presented by Pooranian et al. [55]. The algorithm was heuristically based. The resource allocation mechanism that the author devised is referred to as the "bin packing penalty" and is represented by fog servers. Every virtual machine was used in accordance with time and frequency constraints. An additional policy, known as the "penalty and reward policy," is employed to optimise energy usage.

In order to raise awareness about vehicular networks, [19] suggested an integrated structure. To test the flexibility, they have been turning to the OMNeT++ framework. In order to reconcile local data, a blockchain-based Consensus Sensing (CS) application was created for this study. A unique bio-inspired hybrid algorithm was presented by Rafique et al. [56] for effective resource management in fog computing. According to the volume of incoming requests, the aforementioned work allocated and managed the resources. The major goal of this effort was to schedule the jobs in an efficient manner in order to minimise the average response time and maximise resource utilisation. Task scheduling was handled by the scheduler that was installed in between the fog nodes and the devices. By integrating Modified Particle Swarm Optimisation (MPSO) and Modified Cat Swarm Optimisation (MCSO), the task's inefficient scheduling was resolved. This method was verified, and the outcomes demonstrated that it is more accurate at scheduling the tasks.

The studies listed above have all suggested scheduling algorithms; they have not addressed how user requests change dynamically in the ever-changing cloud-fog environment. Based on an analysis of recent research, it has been determined that the field of cloud computing is the primary focus for studying dynamic resource management. As a result, this paper proposes a novel method for scheduling and resource provisioning that will enable dynamic application management.

Table 2.1

Comparison of Different Scheduling Techniques in Fog Computing

Scheduling Technique	Description	Advantages	Disadvantages	Reference
Reinforcement Learning-Based Scheduling (DRL)	Uses deep reinforcement learning to adaptively allocate resources and schedule tasks based on system dynamics.	Learns optimal policies over time, adapts to changing workloads.	Training is time-consuming and requires large datasets.	Zhang et al., 2022[72]
Improved Firework Algorithm (IFWA)	Optimization algorithm for scheduling tasks in fog with better delay-resource balance.	Handles dynamic task arrival, reduces execution delay.	Tasks cannot be preempted, limiting flexibility.	Wang et al., 2023[73]
Two-phase Scheduling with Deep Learning (TPS-DL)	Combines early classification with reinforcement learning for adaptive scheduling.	Better response time, adapts to workload changes.	High training complexity and resource overhead	Shadroo et al., 2021[74]
First-Come-First-Served (FCFS)	Tasks are scheduled in the order of	Simple to implement, easy to	May lead to long waiting times for short	[Yi et al., 2015][43]

	their arrival.	understand.	tasks if a long task arrives first.	
Round Robin	Each task is assigned a fixed time slot in a cyclic order.	Fair to all tasks, simple to implement.	Context switching overhead can be high.	[Chiang & Zhang, 2016][44]
Priority Scheduling	Tasks are scheduled based on priority levels assigned to them.	High-priority tasks are executed first, which can be beneficial for critical applications.	Lower priority tasks may suffer from starvation.	[Bonomi et al., 2012][45]
Shortest Job Next (SJN)	Tasks with the shortest execution time are scheduled first.	Can minimize the average waiting time.	Requires knowledge of execution time in advance, which may not always be possible.	[Stojmenovic & Wen, 2014][46]
Ant Colony Optimization (ACO)	Bio-inspired algorithm that uses the behavior of ants to find optimal paths for task scheduling.	Can find near-optimal solutions, adaptable to dynamic changes.	Computationally intensive, may require significant time to converge.	[Dastjerdi et al., 2016][47]
Particle Swarm Optimization (PSO)	Optimization technique inspired by social behavior	Can efficiently explore the search space, good for	Requires fine-tuning of parameters, convergence	[Gupta et al., 2016][48]

	of birds flocking or fish schooling, used for task scheduling.	handling dynamic environments.	may not be guaranteed.	
Dynamic Least Load First (DLLF)	Tasks are scheduled to the node with the least current load.	Balances the load effectively, reduces the chances of any single node becoming a bottleneck.	May not always result in the shortest total execution time for all tasks.	[Stojmenovic & Wen, 2014][49]
Genetic Algorithms (GA)	Uses principles of natural selection and genetics for scheduling tasks.	Capable of finding high-quality solutions for complex problems.	Can be computationally expensive, may require significant time to reach an optimal solution.	[Dastjerdi et al., 2016][50])
Heuristic-based Scheduling	Utilizes heuristic methods to make scheduling decisions based on predefined rules or experience.	Can be fast and effective for specific types of tasks or environments.	May not always find the best possible solution, performance is highly dependent on the quality of the heuristics used.	[Gupta et al., 2016][51]

2.2.2 Existing Framework in Fog Computing:

This pioneering paper by Bonomi et al. introduces the concept of fog computing as an extension of cloud computing closer to the edge of the network. It discusses the role of fog computing in handling the massive amounts of data generated by IoT devices, reducing latency, and conserving bandwidth. The paper highlights early use cases in smart grids, connected vehicles, and smart cities [38].

Yi and colleagues provide a comprehensive survey of fog computing, discussing its fundamental concepts, applications, and the issues that need to be addressed. The paper elaborates on the architectural components of fog computing and its potential to support real-time analytics, enhanced security, and improved system scalability [39].

Chiang and Zhang's work explores the synergy between fog computing and IoT. The paper identifies research opportunities and challenges in integrating fog and IoT, such as managing heterogeneous devices, ensuring data security, and developing efficient resource allocation strategies. The authors argue for a collaborative approach to address these challenges [40].

EdgeX Foundry is an open-source initiative aimed at building a common framework for industrial IoT edge computing. The project seeks to standardize the development of IoT solutions across diverse hardware and software environments. The framework's modular design supports scalability and interoperability, making it suitable for various industrial applications [41].

This chapter presents a detailed taxonomy and survey of fog computing, highlighting its distinguishing features, architectural models, and key applications. Mahmud et al. discuss the benefits of fog computing in terms of latency reduction, bandwidth optimization, and enhanced security. They also propose future research directions, including standardization efforts and the development of robust fog ecosystems.[42]

This work delves into the security challenges associated with fog computing. Stojmenovic and Wen analyze potential security threats and propose a set of guidelines for designing secure fog systems. They emphasize the need for robust authentication, encryption, and data integrity mechanisms to protect against cyber-attacks [43].

Hong et al. introduce the concept of "Mobile Fog," a programming model designed to support large-scale IoT applications. The paper discusses how Mobile Fog can facilitate the deployment of distributed applications by leveraging the computational resources of mobile

devices and edge nodes. This approach aims to reduce latency and improve the responsiveness of IoT systems [44].

This exploratory study investigates the potential applications of fog computing in healthcare. Skala et al. discuss how fog computing can enhance patient monitoring, medical data analysis, and emergency response. The study highlights the importance of low latency and high availability in healthcare applications and demonstrates how fog computing can meet these requirements [45].

Varshney's paper focuses on the integration of fog computing with pervasive healthcare systems. The author examines how fog computing can support the real-time processing of health data, improve patient care, and enable remote health monitoring. The paper also discusses the challenges of implementing fog computing in healthcare, such as data privacy and interoperability [46].

Dastjerdi and Buyya's work provides a comprehensive overview of how fog computing can help IoT systems achieve their full potential. The authors discuss the architectural components of fog computing, its benefits, and the challenges that need to be addressed. They also present a case study on smart traffic management to illustrate the practical applications of fog computing [47].

This paper explores the use of container technologies in fog computing for industrial IoT applications. Zhao et al. discuss the advantages of using containers, such as scalability, portability, and resource efficiency. They also propose a deployment and management framework that leverages container orchestration tools like Kubernetes to optimize resource utilization in fog environments [48].

Vaquero and Rodero-Merino provide a comprehensive definition of fog computing, distinguishing it from related paradigms like cloud and edge computing. The paper outlines the key characteristics of fog computing, including its ability to support latency-sensitive applications, distribute data processing closer to the source, and provide enhanced data privacy and security [49]

Table 2.2 Comparison of Cited Works

Title	**Authors**	**Year**	**Focus Area**	**Key Contributions**
Fog computing and its role in the internet of things	Bonomi, F., Milito, R., Zhu, J., & Addepalli, S.[46]	2012	Role of fog computing in IoT	Introduced fog computing as an extension of cloud computing, highlighted early use cases.
A survey of fog computing: Concepts, applications and issues	Yi, S., Li, C., & Li, Q.[44]	2015	Survey of fog computing concepts, applications, and issues	Comprehensive survey of fog computing, discussed architectural components and key applications.
Fog and IoT: An overview of research opportunities	Chiang, M., & Zhang, T [45]	2016	Research opportunities in fog computing and IoT	Explored synergy between fog computing and IoT, identified research challenges and opportunities.
Open source industrial IoT edge platform	EdgeX Foundry[42]	2021	Open-source framework for industrial IoT edge computing	Standardized framework for industrial IoT, supports scalability and interoperability.
Fog computing: A taxonomy, survey and future directions	Mahmud, R., Kotagiri, R., & Buyya, R.[43]	2018	Taxonomy and survey of fog computing	Detailed taxonomy, highlighted benefits, challenges, and future research directions.
The fog computing paradigm: Scenarios and security issues	Stojmenovic, I., & Wen, S.[47]	2014	Security issues in fog computing	Analyzed security threats, proposed guidelines for designing secure fog systems.

Mobile fog: A programming model for large-scale applications on the internet of things	Hong, K., Lillethun, D., Ramachandran, U., Ottenwälder, B., & Koldehofe, B. [36]	2013	Programming model for large-scale IoT applications	Introduced 'Mobile Fog', discussed deployment of distributed applications.
Application of fog computing in healthcare: An exploratory study	Skala, K., et al.[77]	2015	Application of fog computing in healthcare	Investigated potential applications in healthcare, emphasized low latency and high availability.
Pervasive healthcare and fog computing	Varshney, U.[78]	2017	Integration of fog computing with healthcare systems	Examined integration with healthcare, discussed real-time data processing and remote monitoring.
Fog computing: Helping the Internet of Things realize its potential	Dastjerdi, A. V., & Buyya, R. [48]	2016	Potential of fog computing in IoT	Overview of architectural components, case study on smart traffic management.
Deployment and management of container-based fog computing in industrial IoT	Zhao, Z., et al.[75]	2018	Container-based fog computing in industrial IoT	Discussed container technologies, proposed deployment and management framework.

Finding your way in the fog: Towards a comprehensive definition of fog computing	Vaquero, L. M., & Rodero-Merino, L.[76]	2014	Comprehensive definition of fog computing	Provided a comprehensive definition, outlined key characteristics of fog computing.
Smart e-health gateway: Bringing intelligence to internet-of-things based ubiquitous healthcare systems	Rahmani, A. M., Thanigaivelan, N. K., Gia, T. N., Granados, J., Negash, B., Liljeberg, P., & Tenhunen, H.	2018	Smart e-health gateway for IoT-based healthcare	Introduced Smart e-Health Gateway, discussed real-time analytics and remote monitoring.
Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT	Aazam, M., & Huh, E. N.[30]	2014	Dynamic resource estimation and pricing for IoT	Proposed dynamic resource estimation and pricing model for micro data centers.
Dependability evaluation in fog computing for the internet of things applications	Santos, R., Maciel, P., & Matos, R. [17]	2018	Dependability evaluation in fog computing	Presented framework for dependability evaluation, emphasized reliability and availability.

Frameworks have been developed under the fog computing paradigm, including IoT devices and the cloud. Liu et al. introduced a methodology to decrease the delay of resource allocation. This architecture demonstrated vehicular Adhoc networks (VANET) to transmit significant data across communication channels. Resource allocation and job scheduling

issues have been overcome using MU-MIMO channels, where data is segmented into pieces and sent. They analysed a specific application situation and optimised resources by identifying and fixing the issue using a genetic algorithm

Tuli et al. [52] designed a lightweight framework called FogBus for connecting IoT-enabled devices.. The framework was developed to incorporate blockchain technology and an authentication procedure to safeguard sensitive data. The functioning framework was assessed using a finger pulse oximeter for Sleep Apnea diagnosis. FogBus fully enables

distributed application execution. There were no policies in place for real-time resource management and application migration during execution.

Rathee et al. [53] introduced a dependable method utilising the tidal trust algorithm to calculate the Trust Value and Trust Factor (TV/TF) to identify genuine FN and IoT devices accurately. The Social Impact Theory Optimizer (SITO) was utilised on the fog layer to compute trust levels in the suggested framework. They identified the malicious nodes in their research by using specific criteria. The framework underwent testing on several parameters, and a virtual fog environment was created using the NS2 simulator. The study needs to account for the dynamic nature of IoT devices in the suggested framework.

Yigitolglu et al. [114] named a framework they created "foggy." This framework oversees the automatic deployment of IoT applications in fog computing environments. The framework has components such as a container registry, version control server, orchestration server, node, and tool for continuous integration. The developed framework has yet to be utilised for practical IoT applications. Zhang and colleagues (2018) developed the Hierarchical Game Framework to address resource allocation issues in fog computing.

Lin et al. [55] developed a hybrid deep learning framework to enhance the efficiency of manufacturing systems. Visual sensors are included into the proposed framework to identify faulty products and measure the extent of the problem. This approach signifies the decrease in the burden on the cloud layer.

2.3 Real time where Fog computing is Applicable:

Fog computing applies to latency-sensitive applications, including healthcare, emergency services, and cyber-physical systems. Below are some instances of fog computing

applications. Most academics focus on fog computing applications, particularly in health care. Various research on health monitoring, detection, diagnosis, and visualisation have been conducted recently. Cao et al. introduced FAST, a distributed analytic system utilising fog computing to monitor stroke migration by including a fall detection algorithm. The suggested technique has been integrated into a fog-based distributed fall detection system. This strategy distributes the analytical workload across the network by dividing the detection responsibilities between the edge devices and the server.

2.3 Problem Formulation:

Resource management is a critical challenge in fog computing environments, and efficient task scheduling is vital for effective resource utilization. While current research emphasizes task scheduling, it often overlooks the optimal schedulability of these tasks. To address this gap, optimization techniques have been employed to enhance task scheduling. The Modified Marine Predators Algorithm has been implemented to overcome the obstacles associated with task scheduling in fog computing, ensuring better resource management [57][58]. A ranking method was employed to ascertain the number of consecutive iterations needed to surpass the current position. Wang et al. proposed an enhanced firework algorithm aimed at achieving optimal task scheduling in fog computing environments [58].

The previous study used the marine predators algorithm to improve energy efficiency in task scheduling. However, it overlooked the balance between delay and task load, which led to resource wastage in the fog node. Moreover, the algorithm scheduled tasks without considering whether resources were available.

This method of task scheduling with the improved firework algorithm has several limitations, one of which is that tasks cannot be preempted. This restriction decreases the overall efficiency of the approach.

The dynamic nature of the fog node, which changes with varying tasks, is not considered, impacting the effectiveness of the proposed task scheduling method in IoT-based fog computing.

Rafique et al. and Shardoo et al. [59] and [60] managed resources for task execution by addressing inefficient task scheduling with Modified Particle Swarm Optimization (MPSO) and Modified Cat Swarm Optimization (MCSO). This approach allocated and managed resources according to incoming request demands [59]. For resource management, three methods were used: Self Organizing Map (SOM) and autoencoder [60]. The "earliest

deadline first" strategy was applied for task scheduling. There are problems with these approach:

- The bio-inspired hybrid algorithm fails to meet QoS and SLA requirements for effective resource management and task scheduling, resulting in suboptimal system performance.
- The Modified Particle Swarm Optimization (MPSO) used in this method tends to converge too early, especially during the scattering phase, which diminishes the approach's overall efficiency.
- The two-phase scheduling approach effectively organizes tasks, but the random allocation leads to increased overload and retransmissions, which in turn raises the average response time.

Research Objectives:

- To analyze existing energy efficiency-based resource allocation algorithms in Fog Computing environment
- To design a resource management framework for the Fog Computing environment.
- To design the proposed energy efficient based resource allocation algorithm in Fog computing environment.
- To validate the above proposed algorithm and compare with existing work in Fog Computing environment

This chapter delved into the current resource scheduling techniques used in fog computing environments. It reviewed the existing frameworks that have been applied within the fog computing paradigm and evaluated the resource management strategies currently in place at the fog layer. The subsequent chapter will introduce a new resource scheduling framework designed to tackle the issues identified in the problem formulation and achieve the objectives set out in this research.

Chapter 3

Proposed Framework for Energy efficient Framework

The previous chapter provided an in-depth exploration of resource sharing and optimization in the fog computing environment. Through a review of related work, it became evident that while resource has been studied in fog computing, there has been a lack of emphasis on resource optimization for scientific processes. This chapter seeks to fill this gap by presenting the architecture of fog computing for optimizing resources in scientific workflow applications.

To achieve specific goals, related actions known as workflows need to be completed. In the realm of cloud computing, these workflows might include tasks such as data processing, application activation, and provisioning of virtual machines. By optimizing the timing of these activities, companies can reduce costs, enhance productivity, and improve the overall efficiency of their IT operations.

Optimizing workflows can significantly enhance the overall efficiency of IT operations. By automating routine tasks and reducing the need for manual intervention, businesses can free up their IT teams to focus on more critical projects. This not only improves the quality of customer service but also makes companies more adaptable and responsive to changing business needs.

This study's primary goal is to present an optimization framework for scientific workflow design. Utilizing the Bayesian framework and the maximum likelihood technique, the study enhances result accuracy through optimal estimations and predictions. It tackles the complexity of multi-objective optimization problems by integrating a random distribution element, introducing variability into the process. This approach enables the model to explore a broader range of solutions, potentially uncovering more diverse and effective outcomes. Furthermore, the research incorporates multiple heuristic techniques—efficient and effective

problem-solving strategies—to improve the model's ability to navigate complex optimization scenarios. Despite the use of randomness, the study strategically minimizes its impact to ensure the reliability of results and reduce the influence of unpredictable factors.

There are several key reasons to optimize cloud operations, with cost reduction being one of the most significant. By streamlining processes to minimize the time and resources required to complete tasks, businesses can cut expenses. This is especially important for companies that need to scale their computing resources up or down in response to fluctuating workloads.

The suggested methodology considers three primary factors while implementing resource optimization in a fog environment: execution time, computational cost, and energy usage. In certain situations, the processing and storage of a substantial volume of data necessitate the utilization of resources. A significant number of academics prioritize enhancing the performance of fog computing by addressing crucial issues such as privacy, scheduling, security, etc. Fog computing encounters several challenges as it continues to expand, including limited storage capacity, concerns about privacy arising from location awareness, resource overload, increased energy usage, and the need for effective resource management.

In [14], a scheduling strategy called fog Match—based on game theory—was presented. To achieve the lowest possible latency and efficient resource optimization of the corresponding fog nodes, the research work focused on matching the duties of IoT devices to relevant fog nodes. Depending on the need, the aforementioned method introduced both distributed and centralized scheduling. The results showed that in terms of scheduling and better resource management, this work performed better. When fog nodes and IoT devices are matched, resource management is successful.

This research mainly focuses on the resource optimization issue, which means the resources need optimization. Although fog computing improves computational efficiency at the network edge, effective resource optimization continues to pose difficulties that, if unaddressed, may compromise performance in certain scenarios. It efficiently distributes the workload among all the fog resources, considering system requirements. Efficient resource distribution is required in fog computing to enhance the utilization of resources and to provide high-quality services to the users.

In a fog computing environment When it comes to fog resources, a scheduling technique that minimizes an application's makespan but uses a lot of energy is not the best option. When several competing goals need to be met at once, this gets harder. Reducing makespan, for example, while also lowering the amount of energy needed to finish application processing, is difficult. Consequently, to determine the best compromise between these optimization objectives, a biobjective optimization strategy is needed.

The scheduling problem is not well researched for fog-cloud infrastructures, despite having been extensively studied for cloud settings as a single goal or multi-objective optimization problem. In this research work, we first frame the problem as a multi-objective optimization model that takes energy consumption reduction and makespan minimization into account. Given the nature of competition between the two objectives, we employ an adaptive weighted bi-objective cost function. Which of the two criteria—makespan or energy—a user values more highly is indicated by the weight's value. The ultimate goal is to strike the ideal balance between the amount of time it takes to complete an application and the energy used to execute the process.

3.1 EERO: Energy Efficient resource Optimization for scientific workflow application

Workflows are utilized to carry out various experiments. While other resources communicate with one another, a lot of data is transferred. In fog computing, the majority of workflow tasks are performed locally on fog nodes as opposed to being sent to the cloud. Nevertheless, load balancing optimization is required as data transmission between several fog nodes increases to prevent either fog node from having too many jobs or too few duties. As a result, these resources use more energy to complete the jobs, which drives up the hardware cost of fog nodes. Therefore, load balancing can aid in enhancing system performance and lowering the energy and execution time of workflow tasks.

In order to avoid resource overload in the scientific workflow application-based fog computing, this section presented an architecture of load balancing (EERO) for fog computing that reduces cost, execution time, and energy consumption. The suggested EERO model is displayed in Figure 3.1

To enhance energy efficiency in fog computing—particularly for applications utilizing scientific workflows—there is a need for a specialized framework. We introduce EERO (Energy Efficient Resource Optimization) for Fog computing, aimed at minimizing costs, execution time, and energy consumption. As illustrated in Fig. 1, the proposed EERO model

includes a three-layer structure for optimizing resources in an energy-efficient fog architecture. These layers consist of the fog layer, the end-user layer, and the cloud layer. This model retains the essential characteristics of standard fog computing architecture but incorporates an improved fog layer. A detailed description of each layer follows.

End-user Layer: At the network's edge, end users initiate requests that are directed to the fog layer. With the growing demand, scientific workflow applications produce millions of tasks per second. These tasks are first processed before being forwarded to the fog layer for execution. To ensure an efficient distribution of work, we apply the Pareto distribution method. While some tasks are handled within the fog layer, others are sent onward to the cloud layer for processing.

Fog Layer: The fog layer is organized into multiple clusters, each containing a few fog nodes. Each cluster includes a local controller responsible for monitoring fog nodes and maximizing resource utilization. Users connected to the fog layer continuously send requests to these fog nodes, generating a large volume of tasks due to the high number of users. This setup brings connectivity services closer to the data-producing nodes at the most immediate layer. The system comprises physical and virtual sensors, computing nodes, and other components. Within the fog layer, there are small data centers—similar to limited-function clouds—known as nano data centers. These centers have restricted processing and storage capabilities, so only high-priority tasks are handled locally, while others are sent to the cloud layer for processing.

Cloud Layer: The cloud layer connects with the fog layer to support future data transmission and storage needs. This layer consists of large data centers equipped with extensive networking, storage, and processing capabilities. These data centers provide repository support for lower-priority tasks from nano data centers in the fog layer, allowing them to be stored and accessed
for future use.

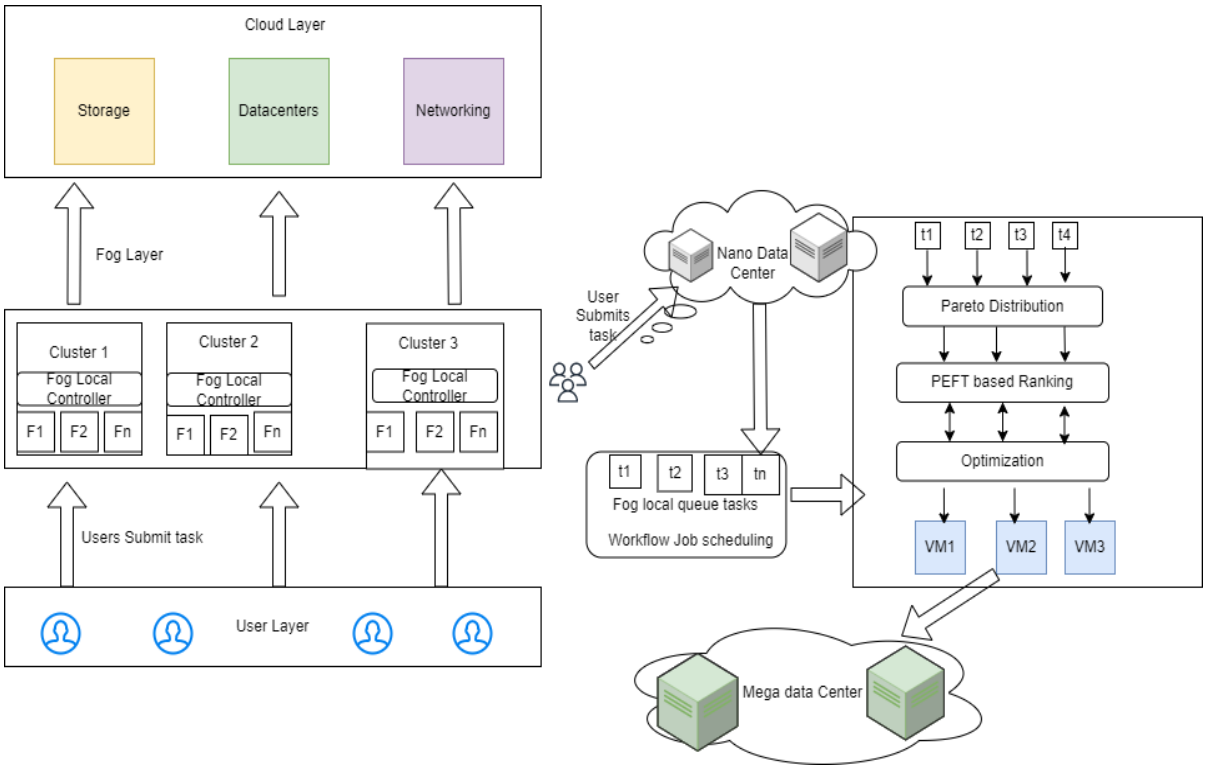


Figure 3.1 Energy Efficient Resource Optimization Model

3.1.1 Operating modules of EERO

In terms of operation, the proposed model is divided into three key modules: the optimization module, the pre-processing module, and the parameter analysis module, as shown in Fig. 2. A detailed description of the updated process for each module is provided below.

3.1.1 Initial Processing or Pre-Processing module: The Workflow Management System (WFMS) is utilized to break down workflows into a series of activities, enabling their automated and efficient execution. This system allows users to design and review workflows, set budgets, specify time constraints, and choose preferred working conditions. To ensure tasks are allocated effectively and stay within budget and deadlines, we employ the Pareto distribution. After reviewing and implementing these parameters within the defined limits, the WFMS assesses dependencies and sends completed tasks to the scheduler via the task dispatcher.

3.1.2 Optimization Process or module: This method provides the user with full transparency regarding the services they received while completing various tasks. When all nodes have the necessary resources, the tasks assigned to the fog nodes are successfully completed. However, if some tasks lack resources and the fog nodes are still underutilized, resource

optimization becomes necessary. To address this, we apply the PEFT ranking algorithm to the available tasks.

3.1.3 Analysis Module for Parameters: After resource optimization, an analysis is conducted on key parameters, including cost, energy usage, and execution time. If the evaluation reveals that further optimization is needed, the tasks are sent back to the optimization module for rescheduling.

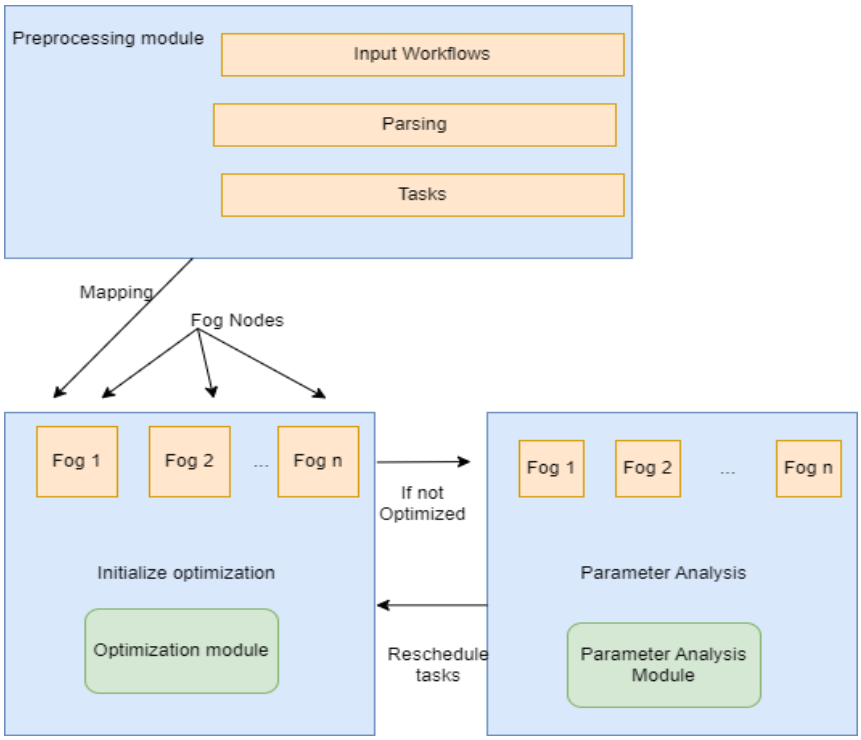


Figure 3.2Operating module of EERO

The working method is explained in the fig 3.1.2

Key components of our methodology include:

- Workflow Management and Parsing: Utilizing the Workflow Management System (WFMS), we parse complex workflows into manageable tasks, facilitating efficient execution and resource allocation.
- Pareto Distribution: By applying the Pareto principle, we prioritize critical tasks, ensuring optimal use of resources and balancing the load across the fog network.
- PEFT Ranking Algorithm: The Predict Earliest Finish Time (PEFT) algorithm ranks tasks based on their dependencies and execution times, allowing for more effective scheduling.

- Genetic Algorithm and Bayesian Optimization: These techniques are employed to refine task scheduling, finding the optimal configuration that balances energy consumption and execution time.
- Adaptive Re-Optimization: The system continuously monitors execution results, dynamically adjusting schedules to address any inefficiencies or changes in workload demands.

The implementation of these strategies within the EERO framework has demonstrated significant improvements in the overall performance of fog computing environments. By reducing the energy consumption and execution times, our model not only enhances the efficiency of scientific workflows but also contributes to the sustainability of computational infrastructures.

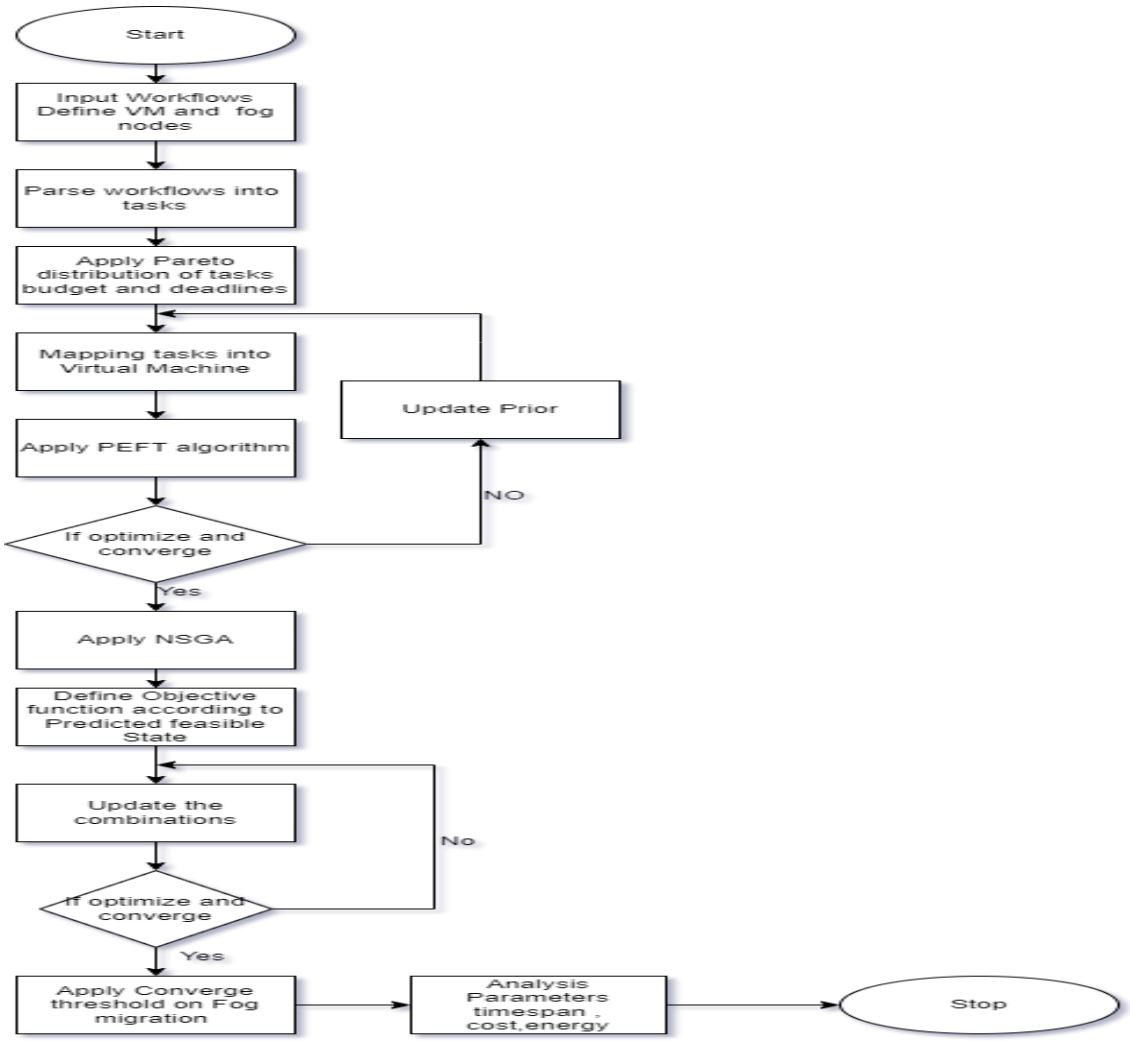


Figure 3.1.2 Working methodology of EERO

3.1.2 Assignment of a workflow task:

The process for distributing tasks among fog nodes is detailed here. The workflow organizer collects tasks generated by users, places them into a queue, and holds them until processing resources are ready. Remote users submit their workloads to fog nodes for execution. The workflow scheduler prioritizes these tasks and assigns them to the fog's local controller. Once resources become available, tasks are sent for execution, and the task scheduler is updated on their status afterward. This approach helps reduce the load on the fog layer by allocating tasks as resources free up. The fog cluster's coordinator node oversees load distribution across virtual machines, shifting tasks from overloaded virtual machines to idle ones to balance the load effectively.

3.1.3 Proposed work flow model:

A Directed Acyclic Graph (DAG) can be represented by a set of vertices (V1, V2, V3…, Vn), with edges (E1, E2, E3…, En) defining the processes in fog computing. Workflows in fog computing can be considered NP-complete problems. In this context, vertices symbolize the tasks assigned to virtual machines (VMs), labeled as "VM1, VM2, VM3…, VMn," while edges denote the interactions between tasks T, such as "T1, T2, T3…, Tn." Workflow weights are assigned to edges by specifying computation and communication times for each task. Resources, represented as "R1, R2, R3…, Rn" within the fog and cloud layers, are allocated to these tasks. This section presents models for time, cost, energy, makespan, and objective functions in fog computing processes.

Table 3.1 Notations

Notation	Description
Vertices	V
Edge	E
Virtual machine	VM
Tasks	T
Resource	R
Total time	T_{ti}
Time required to receive a task	T_R
Time required in the processing of a task	T_P
Waiting time for a task	T_w
Total Cost	T_{co}
resource's ready time	RR_n
fitness function	∂
Optimize parameter	λ
Execution Time	ET
Execution Cost	EC
bjective function	*f(p)*

Time: During workflow execution, a numerical solution can either continue with the current execution phase or reschedule the remaining tasks by assessing the available execution time. Key factors such as job dependencies, task variability, and computational capacity should be accounted for when estimating the workflow's execution duration [16]. An important aspect of scientific workflows is that some runtime components are designed stochastically, adding unpredictability to execution estimates. Execution time in workflows is calculated as the total duration from the start to the completion of a process, factoring in any waiting periods, such as time spent awaiting resources or the completion of other tasks.

$$T_{ti} = \sum_{x=1}^{VMx} T_{Re} + \sum_{x=1}^{VMx} Tp + \sum_{x=1}^{VMx} Tw \quad (3.1)$$

T_{ti} = Total time

T_{Re} = Time required to receive a task

T_P = Time required in the processing of a task

T_w= Waiting time for a task

where VMx indicates how many virtual machines are there overall.

Cost: All cost metrics in this study are **unit less** and represent **normalized values** between 0 and 1 for theoretical comparison. Actual values depend on predefined simulation weights for resource use, migration, and memory. In the execution of scientific procedures, both the cost factor (CF) and the movement factor (MF) are considered. MF represents the ratio of expenses incurred during task execution, factoring in migration and virtual machine (VM) costs. CF is calculated as the ratio of the total process cost to the combined cost of the VM and data center, adjusted by the amount of memory utilized by the task.

$$T_{Co\ \ (Total\ Cost)} = (MF+CF)/2 \quad (3.2)$$

Where MF defines the Movement factor and CF defines the Cost factor

$$MF = \frac{1}{Total\ number\ of\ hosts\ in\ data\ center} \sum_{x=1}^{VMx} \frac{Number\ \ of\ \ migration}{Used\ VM} \quad (3.3)$$

$$CF = \sum_{x=1}^{VMx} \frac{Cost\ to\ process * memory\ of\ tasks}{VM * Data\ \ Center} \quad (3.4)$$

where VMx indicates how many virtual machines are there overall in the system. It is possible to calculate the overall cost of a task that is on time and a task that is late.

Actual Cost = Cost of Underlined tasks + Cost of tasks which has crossed the deadline (3.5)

Energy: Energy is calculated as the total of all instances' movement factor, time, and cost factor. The following equation depicts how much energy the fog environment uses when running operations.

Energy = $\sum_{x=1}^{}(T_{ti} + MF + CF)$* Number of instances (3.6)

The three terms Tti, MF, and CF stand for total time, movement, and cost, respectively

Makespan: It represents the total time required to complete all assigned tasks using the available resources. To estimate this, we use the **Expected Time to Compute (ETC) matrix**, where T_j refers to a specific task and Rj indicates a particular resource. Efficient task scheduling aims to minimize the makespan by balancing the workload across resources.

In this approach, tasks are assigned in a way that avoids overloading any single resource, helping to reduce the total completion time. The **completion time** C for a given task on a resource is calculated as

$$C=RR_n+ER_n$$

Here, RRn is the ready time of resource n, and ERn is the execution time of the task on that resource. Once all completion times are computed, the **makespan (MS)** is defined as the maximum value among them

$$MS=\max(C(T_j,R_n)) \quad (3.7)$$

Objective function: The objective of this study can be outlined using the previously established models for makespan, cost, energy, and time.

$$f(p)= \alpha * (T_{ti}+T_{co}+ E + MS) \quad (3.8)$$

In this model, the objective function f(p) aims to be minimized to achieve the best results. Upon optimization by the algorithm, the fitness value is obtained. The parameters are represented as follows: total cost Tc, energy consumption E , total time Tt , and makespan MS

3.2 Optimization method used:

Utilizing EERO optimization techniques, the main goal is to reduce energy usage. The proposed architecture is divided into four parts

A. Parsing of Workflows
B. Optimize the ranking
C. Optimize the task scheduling
D. Analysis the parameter

We begin by parsing workflows and assigning an optimal ranking. The first step involves identifying the ideal Pareto front, followed by applying a PEFT-based ranking within that region. Once ranked, we analyze the probability distribution correlation of these task ranks and optimize through a Bayesian approach. The Pareto front, comprising nondominated solutions, represents the best options if no goal can be improved without compromising another. Alternatively, a solution x* is considered dominated by another solution x only if x is equal to or better than x* across all objectives. Given that rankings are interdependent, previous step data is utilized to create an efficient task mapping. Consequently, NSGA-II is applied to achieve multi-objective optimization using equation 1, monitor resource usage, and establish an optimal scheduling threshold for virtual machines. Bayesian optimization is particularly employed to locate the global minimum with minimal iterations, providing an effective framework for addressing similar challenges. So, parse the workflows as per the parent-child relationship and the specified order, although many tasks will appear in the series. On the same level, we go to the next phase and assign an optimal ranking.

Fitness function: $F = \partial$ (ET +EC+E) + $\lambda(ET + EC + E)$ …. (3.9)

∂ =learning parameter
λ =optimize parameter
ET=execution time
EC=execution cost
E=energy

To start, initialize N and W to represent the initial values for the number of fog nodes and workflows, respectively. Fog nodes are transformed into parser trees once the nodes and workflows are established. After creating parsing trees for fog nodes, computationally intensive workflow jobs are divided into smaller task components, extracted from workflows, and assigned to fog nodes. The first step involves identifying the optimal Pareto front, which provides a solution to the multi-objective optimization problem. This Pareto front represents a set of optimal, non-dominant options.

Algorithm 3.1: Optimal Pareto Front Selection

Input: Set of tasks with dependencies based on Time and Energy
Output: Optimized Pareto Front
1. D ← Identify task dependencies based on Time and Energy
2. Initialize:
iteration_count ← 0
MAX_ITER ← N
converged ← False
previous_pareto_front ← ∅
ε ← small threshold value (e.g., 1e-3)
3. While (NOT converged AND iteration_count < MAX_ITER):
a. Apply Dominate() using fitness function eq(3.9)
b. current_pareto_front ← Update Pareto Front
c. ΔFront ← compute_distance(previous_pareto_front, current_pareto_front)
d. If ΔFront < ε then
converged ← True
Break
e. previous_pareto_front ← current_pareto_front
f. iteration_count ← iteration_count + 1
4. Return Final Pareto Front

Algorithm 3.2: PEFT-Based Task Ranking

Algorithm: Task Ranking Based on Dependency and Dominance
Input: Pareto Front space
Output: Task ranking according to dependency and dominance
1. D ← Extract task dependencies from Pareto Front
2. Initialize:
iteration_count ← 0
MAX_ITER ← N
converged ← False
previous_rankings ← ∅
ε ← small threshold (e.g., 0)
3. While (NOT converged AND iteration_count < MAX_ITER):
a. For each pair of tasks (t_i, t_j) in Pareto Front:
i. If $t_i > t_j$ (in terms of dependency or priority):
- Apply: Dominate(t_i) > Dominate(t_j)
- Update task rankings accordingly
b. current_rankings ← updated task rankings
c. ΔRank ← compute_ranking_difference(previous_rankings, current_rankings)
d. If ΔRank ≤ ε then
converged ← True
Break
e. previous_rankings ← current_rankings
f. iteration_count ← iteration_count + 1
4. Assign final task rankings based on updated Pareto region
5. Proceed to objective-based task scheduling

Algorithm 3.3: Bayesian Optimization for Task Ranking

Input: DAG (workflows) with PEFT Ranking
Output: Optimized Task Mapping and Scheduling
1. Initialize:
iteration_count ← 0
MAX_ITER ← N
converged ← False
improvement_threshold ε ← small value (e.g., 1e-3)
convergence_counter ← 0
patience ← P (e.g., 3)
2. While (NOT converged AND iteration_count < MAX_ITER):

a. For each task:
i. Compute task execution time and energy at each fog node
ii. Predict expected improvement EI(x) using Bayes: $\mu(x\|D)$, $\sigma(x\|D)$
iii. Store EI(x) for convergence tracking
b. max_EI ← max(EI(x) for all tasks)
c. If max_EI > ε then
Apply BayesOptimize()
convergence_counter ← 0
Else
convergence_counter ← convergence_counter + 1
d. If convergence_counter ≥ patience then
converged ← True
Break
e. iteration_count ← iteration_count + 1
3. Output: Final optimized task ranking and assignment

Workflows Parsing :The DAG workflow design reflects a parent-child relationship. If there's an edge from Parent I to Child J within the DAG, it indicates that Child J is the successor of Parent I. Due to task precedence constraints, Child J can only start after Parent I has completed and passed the necessary information. Thus, workflows should be parsed according to this parent-child relationship and the specified sequence, even when multiple tasks are involved. We then proceed to determine the optimal order of tasks at the same level.

Ranking Optimization :This section is divided into three parts. First, tasks are ranked within the optimized space based on our three research objectives: cost, energy, and time. The initial step identifies the Pareto front, and the second phase ranks the region using PEFT. Next, we establish the probability distribution relationship between these task ranks and the overall process ranking, then apply a Bayesian optimization technique to refine it further. Optimizing workflows can significantly enhance IT operations' overall efficiency. By automating routine tasks and reducing manual intervention, businesses can allow their IT staff to focus on more critical projects. This shift can improve customer service quality and make companies more adaptable and responsive to changing business needs.

Pareto Front:

We assume that all objectives are to be minimized, as any maximization problem can be transformed into a minimization problem. The Pareto set consists of a group of non-dominant solutions representing trade-offs among the objectives. The values of these solutions form the Pareto frontier, which is a powerful tool for identifying preferences and supporting decision-making.

PEFT Ordering

Predict the Earliest Finish Time (PEFT) is a scheduling technique designed for use with a limited number of heterogeneous processors. The algorithm operates in two stages: Task Prioritization, which determines the order of task execution, and Processor Selection, which identifies the most suitable processor for executing each task.

3.3 Workflow of the Algorithm:

1. **Initialize System**: Initializes system parameters, fog nodes, and cloud nodes.
2. **EERO_Model**: Main algorithm that processes workflows, optimizes task scheduling, and assigns tasks to fog nodes.
3. **WFMS_parse_workflow**: Parses workflows into individual tasks.
4. **Pareto_distribution**: Distributes tasks based on the Pareto principle.
5. **PEFT_ranking**: Ranks tasks using the PEFT algorithm.
6. **GA_Bayesian_optimization**: Uses genetic algorithm and Bayesian optimization to find the best task scheduling configuration.
7. **Assign_task_to_fog_node_based_on_optimization**: Assigns tasks to fog nodes based on the optimized schedule.
8. **analyze_execution**: Analyzes execution results from fog nodes.
9. **optimization needed**: Determines if further optimization is needed based on execution results.
10. **re_optimize_schedule**: Re-optimizes the task schedule if necessary.

Initialization

Algorithm: Initialize System

Input: None

Output: Initialized system parameters, fog nodes, cloud nodes

1. Initialize system parameters {cost, energy_consumption, execution_time}

2. For i = 1 to number_of_fog_nodes do

Create fog_node(i) with {RAM, CPU, bandwidth}

End For

3. Create cloud_nodes with {high_capacity_storage, processing_power}

End Algorithm

Algorithm: EERO_Model

Input: workflows, fog_nodes, cloud_nodes

Output: Optimized task scheduling and execution

1. Initialize_System()

2. For each workflow in workflows do

// Step 1: Pre-Processing

tasks = WFMS_parse_workflow(workflow)

distributed_tasks = Pareto_distribution(tasks)

// Step 2: Optimization Process

ranked_tasks = PEFT_ranking(distributed_tasks)

optimized_schedule = GA_Bayesian_optimization(ranked_tasks, fog_nodes)

// Step 3: Execute and Monitor Tasks

```
    For each task in optimized_schedule do
        Assign task to fog_node_based_on_optimization(task, fog_nodes)
    End For

    // Step 4: Analyze Results
    execution_results = analyze_execution(fog_nodes)
    If optimization_needed(execution_results) then
       re_optimized_schedule = re_optimize_schedule(optimized_schedule, execution_results)
        Assign re_optimized_schedule to fog_nodes
    End If
  End For
End Algorithm

Algorithm: WFMS_parse_workflow
Input: workflow
Output: tasks

1. Split workflow into tasks
2. Return tasks
End Algorithm
Algorithm: Pareto_distribution
Input: tasks
Output: distributed_tasks
```

1. Distribute tasks based on Pareto principle

2. Return distributed_tasks

End Algorithm

Algorithm: PEFT_ranking

Input: tasks

Output: ranked_tasks

1. Compute Optimistic Cost Table (OCT)

2. For each task in tasks do

    Calculate earliest_finish_time(task)

  End For

3. Sort tasks by earliest_finish_time

4. Return ranked_tasks

End Algorithm

Algorithm: GA_Bayesian_optimization

Input: ranked_tasks, fog_nodes

Output: optimized_schedule

1. Initialize population with ranked_tasks and fog_nodes

2. For generation = 1 to max_generations do

    // Selection

    selected_individuals = SELECTION(population)

    // Crossover

    offspring = CROSSOVER(selected_individuals)

```
    // Mutation
    mutated_offspring = MUTATION(offspring)

    // Evaluate fitness
    For each individual in mutated_offspring do
        fitness = EVALUATE_fitness(individual, system_parameters)
        Update_population(population, individual, fitness)
    End For

    // Bayesian Optimization
    optimized_individual = Bayesian_optimization(population)
    Update_population_with_optimized_individual(population, optimized_individual)
  End For
3. Return best_individual_from_population(population)
End Algorithm

Algorithm: Assign_task_to_fog_node_based_on_optimization
Input: task, fog_nodes
Output: None

1. Find optimal_fog_node for task based on optimization
2. Assign task to optimal_fog_node
End Algorithm
```

Algorithm: analyze_execution

Input: fog_nodes

Output: execution_results

1. Collect execution data from fog_nodes

2. Return execution_results

End Algorithm

Algorithm: optimization_needed

Input: execution_results

Output: Boolean

1. If execution_results not meeting_thresholds then

    Return True

  Else

    Return False

End Algorithm

Algorithm: re_optimize_schedule

Input: optimized_schedule, execution_results

Output: re_optimized_schedule

1. Re-optimize schedule based on execution_results

2. Return re_optimized_schedule

End Algorithm

Conclusion:

In this chapter, we introduced an innovative framework designed to enhance the efficiency and optimization of resources in fog computing environments, particularly for scientific workflow applications. Our proposed Energy Efficient Resource Optimization (EERO) model offers a structured approach to managing and distributing computational tasks, emphasizing energy conservation, cost reduction, and minimizing execution times.

The EERO model incorporates a multi-layered architecture comprising the end-user layer, fog layer, and cloud layer. This hierarchical structure ensures that tasks are processed efficiently at the edge of the network, leveraging the capabilities of fog nodes to handle local computational demands while offloading more intensive tasks to the cloud as needed. This strategic distribution significantly reduces latency and energy consumption, addressing the core challenges of fog computing.

In conclusion, the EERO model presents a robust solution to the challenges of resource optimization in fog computing. Its adaptive, multi-layered approach ensures that scientific workflow applications are executed with maximum efficiency, paving the way for future advancements in fog computing technologies. The integration of heuristic and probabilistic techniques within the framework underscores the potential for continued innovation in this field, promising more resilient and energy-efficient computing environments.

Chapter 4

Energy Efficient Resource Optimization algorithm for scientific workflows in Fog Computing

4.1 Resource optimization algorithm

The previous chapter introduced a framework focused on resource utilization for scientific workflows. This chapter presents an energy-efficient resource optimization algorithm tailored for scientific workflow applications. The simultaneous data transmission from numerous smart device users leads to resource shortages. Often, some resources are fully utilized while others in the fog layer remain idle, resulting in wasted resources and power. Optimizing resources in the fog computing layer is challenging, as it aims to minimize cost and energy consumption. Load imbalance in the fog layer also wastes bandwidth, reducing throughput and increasing user response time. These issues arise from the constrained environment and limited resource availability.

The EERO algorithm is designed to improve the efficiency of fog computing by balancing the load and optimizing the scheduling of tasks. It integrates several optimization techniques to achieve this goal. The main components of the EERO algorithm include:

1. **Pre-Processing Module**: This module uses the Workflow Management System (WFMS) to split workflows into a collection of tasks. Tasks are then distributed based on the Pareto distribution to ensure they are within budget and deadline constraints.
2. **Optimization Module**: This module applies the PEFT (Pareto Efficient Task) ranking algorithm to rank tasks and uses Bayesian optimization to find the optimal task scheduling.
3. **Parameter Analysis Module**: After optimization, this module analyzes parameters like cost, energy consumption, and execution time. If further optimization is needed, tasks are returned to the optimization module.

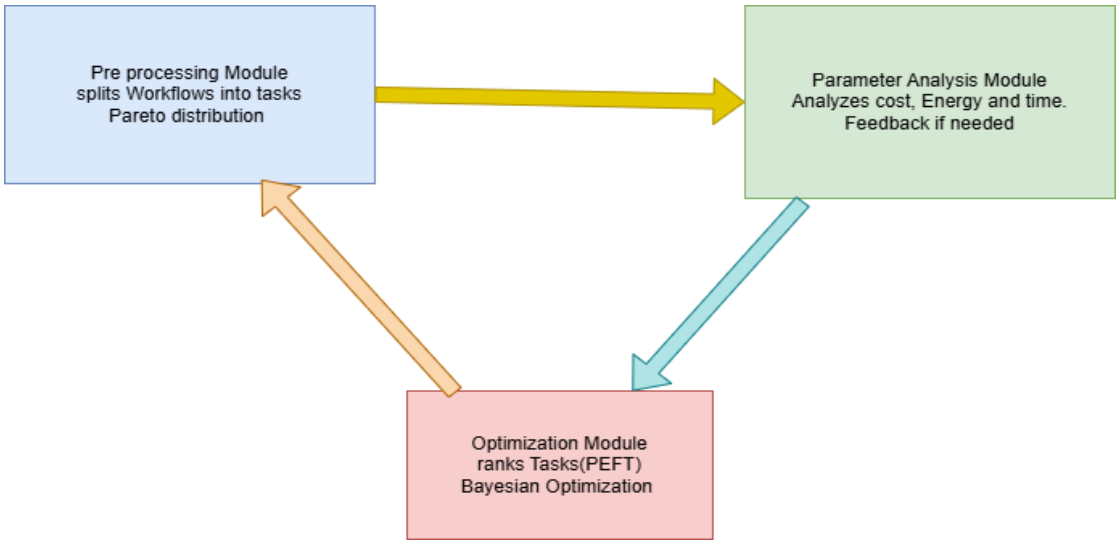


Figure 4.1 Working of EERO

Fog computing is gaining popularity in the Internet of Things (IoT) world. Instead of relying only on cloud datacentres for computing and storing IoT data, fog computing provides local storage and processing right where users need it. This makes IoT more efficient and accessible.

Deciding whether to run applications in the fog layer or the cloud is important for maintaining service quality. To manage this, a cloud-fog scheduler is used to ensure tasks are processed without delays.

Load balancing is key to keeping fog computing systems running smoothly. However, because the fog environment is spread out and has many users, balancing the load can be tricky. With more users, the load fluctuates, making it hard to distribute work evenly. To make the best use of resources in a fog environment, it's important to spread the load across all available virtual machines (VMs) to avoid overloading or underutilizing them.

Scientific workflows are data-intensive applications that handle distributed data sources and complex computations across various fields like astronomy, engineering, and bioinformatics. In distributed environments such as fog computing, numerous sensors and experimental processes produce large volumes of data that must be collected and processed within specific time constraints. Fog computing utilizes geographically distributed resources to manage and process this data efficiently.

Despite its advantages over cloud computing, fog computing faces several challenges. One significant challenge is balancing the load during the execution of scientific workflow tasks

in a complex resource environment. These tasks require real-time implementation, but the substantial data volume can overload fog computing resources. Therefore, it is essential to evenly distribute the data across available resources to ensure real-time processing. Proper distribution of tasks helps in efficient resource utilization, saving both energy and execution time[61]. Scientific workflows are complex, often considered NP-complete problems, involving a series of computational tasks for various scientific applications. These workflows are typically represented as Directed Acyclic Graphs (DAGs), which consist of vertices (V1, V2, V3, ..., Vn) and edges (E1, E2, E3, ..., En). The vertices symbolize different workflow tasks that are assigned to corresponding virtual machines (VM1, VM2, VM3, ..., VMn), while the edges denote the communication between tasks (T1, T2, T3, ..., Tn). Essentially, a DAG is depicted as a tree structure with nodes and connecting edges, where these edges are weighted based on communication and computation time. Various types of workflows can be executed using fog computing, leveraging this DAG representation for efficient task management.

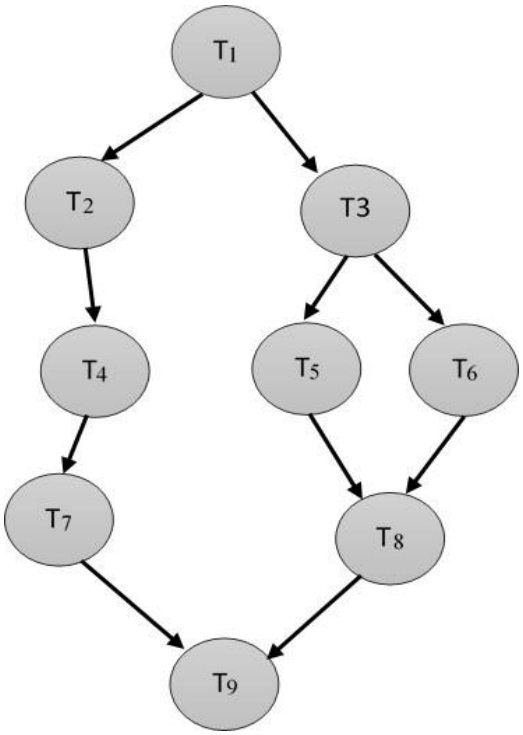


Fig: 4.2 Example of workflow

A Directed Acyclic Graph (DAG) is a fundamental structure used in scientific workflows to represent and manage a series of computational tasks. Here are some key points explaining its use:

1. **Representation of Tasks and Dependencies**:

    - **Vertices**: Each node (or vertex) in a DAG represents an individual task in the workflow. These tasks could be any computational step required by a scientific application.
    - **Edges**: The directed edges between nodes indicate dependencies between tasks. An edge from task A to task B signifies that task B cannot start until task A has been completed.

2. **Execution Order**:

    - The acyclic nature of the graph ensures that there are no circular dependencies, which means the tasks can be scheduled and executed in a specific order. This order respects the dependencies defined by the edges.

3. **Parallel Processing**:

    - DAGs enable the identification of independent tasks that can be executed in parallel. Tasks that do not have direct or indirect dependencies on each other can be processed simultaneously, optimizing resource usage and reducing execution time.

4. **Resource Allocation**:

    - In distributed computing environments, such as fog computing, tasks represented by nodes can be mapped to various virtual machines (VMs) or other computational resources. This mapping helps in effectively utilizing available resources.

5. **Load Balancing**:

- DAGs help in distributing the workload evenly across different resources. By analyzing the graph, the system can allocate tasks to prevent overloading any single resource and ensure efficient task execution.

6. **Performance Optimization**:

   - Weights can be assigned to the edges representing the communication or computation time between tasks. This information helps in optimizing the workflow by minimizing data transfer times and balancing computation loads.

7. **Flexibility and Scalability**:

   - DAGs offer a flexible structure that can be easily modified to accommodate changes in the workflow. New tasks can be added, or existing tasks can be removed or altered without disrupting the overall structure.
   - They also support scalability, as tasks can be distributed across various resources in a geographically distributed environment, like fog or cloud computing.

8. **Error Handling and Recovery**:

   - In case of a failure in one of the tasks, DAGs facilitate error handling and recovery by allowing the system to identify the failed task and re-execute it or take corrective measures without affecting the entire workflow.

By representing scientific workflows as DAGs, complex computational processes can be systematically managed, scheduled, and executed, ensuring efficiency, reliability, and optimal resource utilization.

Examples of workflows from different domains, showcasing how tasks are structured, managed, and executed using a workflow system:

1. Bioinformatics Workflow

- **DNA Sequencing**: A common workflow in bioinformatics involves the sequencing of DNA samples.
    - **Tasks**: Sample preparation, sequencing, data cleaning, alignment of sequences, identification of genetic variants, annotation of variants, and reporting.
    - **Tools**: Various software tools like FASTQC for quality control, BWA for sequence alignment, GATK for variant calling, and custom scripts for data processing.

2. Astronomy Workflow

- **Image Processing**: Processing data from telescopes to generate usable astronomical images.
- **Tasks**: Data acquisition, calibration (removing noise and errors), alignment of images from multiple exposures, stacking (combining images to enhance signal), and final image enhancement.
- **Tools**: Software like IRAF for image processing, DS9 for visualization, and custom scripts for data handling.

3. Engineering Workflow

Finite Element Analysis (FEA): Simulating physical phenomena using computational models.

- **Tasks**: Pre-processing (defining geometry, material properties, boundary conditions), meshing (dividing the geometry into smaller elements), solving (running simulations), and post-processing (analyzing results and visualizing data).
- **Tools**: Software like ANSYS or Abaqus for simulation, and MATLAB or Python for custom post-processing scripts.

4. Business Process Workflow

- **Order Processing**: Managing customer orders in a retail or manufacturing environment.
    - **Tasks**: Order entry, payment processing, inventory check, order fulfilment, shipping, and customer notification.
    - **Tools**: ERP systems like SAP or Oracle, CRM tools for customer management, and custom software for specific process steps.

5. Data Science Workflow

- **Machine Learning Model Development**: Creating predictive models from data.

    - **Tasks**: Data collection, data cleaning, exploratory data analysis, feature engineering, model training, model validation, and deployment.
    - **Tools**: Programming languages like Python or R, libraries such as pandas, scikit-learn, TensorFlow, and cloud platforms like AWS or Azure for deployment.

6. Healthcare Workflow

- **Patient Management**: Coordinating patient care in a hospital.
    - **Tasks**: Patient registration, appointment scheduling, medical examination, diagnostics (lab tests, imaging), treatment planning, and follow-up.
    - **Tools**: Electronic Health Record (EHR) systems, medical imaging software, and custom hospital management software.

7. Media Production Workflow

- **Video Production**: Creating a film or a video segment.
    - **Tasks**: Scriptwriting, storyboarding, shooting, editing, visual effects, sound editing, and final rendering.
    - **Tools**: Software like Adobe Premiere Pro, Final Cut Pro, After Effects for visual effects, and Audition for sound editing.

Resource Optimization based Workflow Execution model for Fog Computing:

This section introduces a workflow execution model tailored for fog computing environments, focusing on optimizing resources. Fog computing, when handling extensive computational tasks, encounters challenges such as scheduling loads, balancing those loads, and utilizing resources efficiently. Our proposed solution aims to improve resource utilization and decrease energy consumption in fog nodes.

The first layer is the end-user layer, where users generate numerous workflow tasks. These tasks are temporarily stored in a workflow container before being assigned to the workflow scheduler. Here's how the process works:

1. The workflow container submits tasks to the workflow scheduler in the order they arrive.
2. The workflow scheduler has a queue where tasks wait for resources. Tasks enter the queue at the back and are removed from the front.
3. As resources become available, tasks are taken from the queue and assigned to the central controller in the fog layer.

Fog Layer:

The second layer of the workflow execution model is the fog layer, which consists of various fog clusters containing multiple fog nodes. This layer also has a central controller that manages the fog clusters. The central controller checks for available nodes in each cluster and assigns tasks to those available nodes.

Fog Layer Execution: Load Balancing and Task Assignment

Once the optimized task schedules have been generated, the central controller in the fog layer takes charge of executing the tasks. It receives workflow tasks from the workflow scheduler and employs a load balancer to continuously monitor all fog nodes across the distributed fog clusters. To ensure efficient task distribution, the system uses the PSW-Fog clustering-based load balancing method, which evenly assigns tasks among available nodes based on real-time resource availability.

Each fog cluster comprises multiple fog nodes, and each node hosts several virtual machines (VMs) responsible for executing the assigned tasks. High-priority tasks are processed

immediately on the most suitable VMs, while lower-priority tasks may be queued or offloaded to the cloud layer for further handling if local resources are insufficient.

The fog layer emphasizes real-time processing, aiming to minimize latency and optimize local resource utilization. Once tasks are executed, users receive responses directly from the fog layer, improving response time and reducing reliance on the cloud.

Cloud Layer Execution: Extended Processing and Storage

The cloud layer represents the third and final tier of the workflow execution model. It consists of large-scale data centres equipped with vast computational, networking, and storage resources. This layer serves as a backup and support infrastructure for handling tasks that exceed the capabilities of the fog layer.

Once tasks are executed in the fog layer, the results are returned to users. However, if additional computation, long-term storage, or batch processing is required, those tasks are escalated to the cloud layer. The cloud handles such overflow tasks with greater processing power, albeit with higher latency. The objective of the proposed workflow execution model is to minimize execution time within the fog layer, thereby reducing the burden on cloud resources and improving overall system efficiency.

4.3 Efficient Resource Optimization

The optimization of scientific workflows in cloud computing environments presents significant challenges due to the conflicting objectives of minimizing execution time, energy consumption, and costs while maintaining quality of service (QoS) standards. This research work proposes a novel approach using a multi-objective genetic algorithm to address these challenges effectively. The algorithm leverages a combination of heuristic and meta-heuristic techniques, including Predict the Earliest Finish Time (PEFT) and Bayesian optimization, to enhance task scheduling efficiency.

The core of the proposed method is a multi-objective genetic algorithm that constructs a Pareto front to identify non-dominated solutions, providing a balanced trade-off among different optimization criteria. The PEFT heuristic predicts the earliest completion times for tasks, allowing for more efficient scheduling. Additionally, Bayesian optimization is employed to improve the reliability and convergence speed of the algorithm by incorporating probabilistic models into the decision-making process.

4.3.1 Optimization approach used in our proposed algorithm:

This research work proposes a multi-objective genetic algorithm to address the complex problem of optimizing scientific workflows in cloud computing environments. This method is designed to simultaneously minimize multiple conflicting objectives such as execution time, energy consumption, and cost, while ensuring the Quality of Service (QoS) standards.

Key Components of the Algorithm:

In the methodology, the first step is workflow parsing, which involves understanding and organizing the tasks based on their dependencies. This is crucial for ensuring that tasks are

executed in the correct sequence, respecting the dependencies inherent in the workflow. The process is outlined as follows:

1. Directed Acyclic Graph (DAG): The workflow is represented as a DAG, where each node represents a task and each edge represents a dependency between tasks. The DAG ensures there are no cycles, meaning that there is a clear start and end point for the workflow.
2. Parent-Child Relationship:
    - Parent Task: A task that must be completed before another task can begin.
    - Child Task: A task that depends on the completion of a parent task.

3. Parsing Process:
    - Identification of Dependencies: Each task's dependencies are identified based on the edges in the DAG. If there is an edge from Task A to Task B, Task B is considered a child of Task A.
    - Execution Order: The tasks are then arranged in a sequence that respects these dependencies. A task can only begin execution once all its parent tasks have been completed.
    - Level Assignment: Tasks are assigned levels based on their position in the DAG. Tasks with no parents are at level 0, their children are at level 1, and so on. This helps in organizing the tasks for subsequent ranking and scheduling processes.

This parsing ensures that the workflow's logical structure is maintained and that all dependencies are respected during execution. By correctly parsing the workflow into its parent-child relationships, the methodology sets a foundation for efficient scheduling and optimization, ensuring that no task is executed before its prerequisites are satisfied.

Next in the context of optimizing scientific workflows in cloud environments, handling multiple conflicting objectives is crucial. Objectives such as minimizing execution time, reducing cost, and lowering energy consumption often conflict with one another. For reducing execution time might increase energy consumption or cost. Pareto front optimization provides an effective means to navigate these trade-offs by identifying a set of optimal solutions that balance the different objectives. The Pareto front is a concept used in multi-objective optimization to identify a set of non-dominated solutions, where no single solution is superior to the others in all objectives. This allows for a balanced trade-off among the different optimization criteria. This process can be detailed as follows:

Consider a multi-objective optimization problem with k objective functions

$$f1(x), f2(x), \dots, fk(x).$$

The goal is to minimize these functions simultaneously:

$$minx \in (x) = [f1(x), f2(x), \dots, fk(x)]$$

This expression represents a multi-objective optimization problem where you aim to minimize the vector of objective functions f(x) over the feasible solution space X. Each $f_i(x)$ is an individual objective function.

A solution x_1 is said to dominate another solution x_2 if:

$$x_1 \prec x_2 \iff \forall i, f_i(x_1) \leq f_i(x_2) \; and \; \exists j \; such \; that \; f\square(x_1) < f\square(x_2)$$

The Pareto optimal set P^* and Pareto front PF are defined as:

$$P* = \{ x \in X \mid \nexists y \in X such \; that \; y \prec x \}$$

$$PF = \{f(x) \mid x \in P*\}$$

This provides a diverse set of solutions, offering various trade-offs between objectives. This diversity is crucial for decision-makers to choose the most appropriate solution based on specific needs and constraints.

In the context of this study, the Pareto front helps in identifying the optimal scheduling of tasks that balance between minimizing time, cost, and energy consumption.

1. **Predict the Earliest Finish Time (PEFT):**
    - PEFT is a heuristic that estimates the earliest possible completion time for tasks based on their dependencies and the available computing resources.
    - This prediction is crucial for efficient scheduling as it allows the algorithm to prioritize tasks that can be completed earlier, thereby improving overall workflow execution time.

PEFT algorithm calculates the earliest finish time for each task using the following steps:

Initialization:

For each task t_i_ initialize the Earliest Start Time (EST) and Earliest Finish Time (EFT).

For the entry task (a task with no predecessors), the EST is set to zero.

Calculate EST and EFT:

- For each task $t_{i\_}$:
    - Calculate the Earliest Start Time (EST) based on the Earliest Finish Time (EFT) of its predecessors.

      The EST for task tit_iti is given by:

$$EST(ti) = tj \in pred(ti)max(EFT(tj))$$

      where pred(t_i) is the set of predecessor tasks of t_i

Calculate the Earliest Finish Time (EFT) by adding the execution time of the task to its EST:

$$EFT(ti) = EST(ti) + exec\_time(ti)$$

Then we do rank Tasks

- Rank tasks based on their EFT. Tasks with earlier EFTs are given higher priority.

Bayesian Optimization (BO) is used to optimize the ranking process further by predicting the best task orderings and resource assignments. It incorporates probabilistic models to make informed decisions about the scheduling of tasks. This approach helps in refining the search process by focusing on the most promising areas of the solution space, thereby improving the convergence speed and quality of the solutions.

Bayesian Optimization involves the following steps:

Surrogate Model Construction: A probabilistic model (typically a Gaussian Process) is used to approximate the objective function.

Acquisition Function Maximization: An acquisition function, which balances exploration and exploitation, is optimized to decide the next point to evaluate.

Objective Function Evaluation: The true objective function is evaluated at the selected point.

Model Update: The surrogate model is updated with the new data.

Objective Function: Let f(x) be the objective function representing the performance measure (e.g., makespan, cost, energy consumption) that needs to be minimized. Here, x represents the task orderings and resource assignments.

Surrogate Model: A Gaussian Process (GP) is used as the surrogate model. The GP provides a posterior distribution over the objective function f(x) given a set of observed data $D = \{(xi, yi)\} = {}^{n}_{1}$ where $y_{i\_}$ is the observed value of the objective function at x_i

The posterior distribution is given by:$(x) \sim N(\mu(x), \sigma 2(x))$

Where $(\boldsymbol{x})$ is the mean function and $(\boldsymbol{x})$ is the variance function of the GP

Acquisition Function: The acquisition function α(x) is used to determine the next point to evaluate. Common choices include Expected Improvement (EI), Probability of Improvement (PI), and Upper Confidence Bound (UCB).

The Expected Improvement (EI) is defined as:$(x) = E[max(0, f(x+) - f(x))]$

where x+ is the best observed point so far.

Optimization of Acquisition Function:

The next point X_{n+1} to evaluate is chosen by maximizing the acquisition function

$$xn + 1 = \boldsymbol{argmax}\ (x)$$

Heuristic and Ranking Approaches

The algorithm integrates several heuristic and ranking techniques to improve the efficiency of task scheduling:

1. **Efficient Ranking Heuristic:**

- This heuristic ranks tasks based on their importance and urgency. The ranking is used to determine the order in which tasks should be scheduled, ensuring that critical tasks are prioritized.
- The heuristic considers factors such as task dependencies, deadlines, and resource availability to generate an optimal schedule.

2. **Bayesian Approach:**
    - The Bayesian approach is used to incorporate uncertainty and probabilistic reasoning into the scheduling process. It helps in predicting the outcomes of different scheduling decisions and selecting the best course of action.
    - This method enhances the robustness of the scheduling algorithm by accounting for the variability in task execution times and resource performance.

The proposed optimization framework combines the strengths of the genetic algorithm, PEFT heuristic, and Bayesian optimization to tackle the multi-objective optimization problem effectively.

1. **Initialization:**
    - The genetic algorithm starts with an initial population of potential solutions, each representing a possible schedule for the workflow tasks.
    - These initial solutions are generated randomly or based on simple heuristics to cover a diverse range of possible schedules.
2. **Selection:**
    - The selection process involves choosing the most promising solutions from the current population based on their fitness. The fitness is evaluated using the multi-objective criteria of time, cost, and energy consumption.
    - Solutions that lie on the Pareto front are given higher priority as they represent the best trade-offs among the different objectives.
3. **Crossover and Mutation:**
    - The algorithm applies crossover and mutation operators to generate new solutions from the selected ones. Crossover combines parts of two solutions to create a new one, while mutation introduces small changes to a solution to explore the solution space.

- These operators ensure the diversity of the population and help in avoiding local optima.

4. **Evaluation:**
    - The new solutions are evaluated using the PEFT heuristic and Bayesian optimization to estimate their performance in terms of the defined objectives.
    - The evaluation process involves calculating the execution time, cost, and energy consumption for each solution, and updating the Pareto front accordingly.
5. **Iteration:**
    - The algorithm iterates through the selection, crossover, mutation, and evaluation steps until a stopping criterion is met, such as a maximum number of generations or convergence to a stable Pareto front.

CHAPTER 5

RESULTS AND DISCUSSION

This chapter primarily deals with verifying and validating the proposed framework. It outlines the experimental prerequisites and explains the performance evaluation metrics. The EERO framework was implemented using the iFogSim toolkit, and the results were compared to algorithms like ABC, ACO, Tabu Search, and GWO. The analysis of the result graphs shows that the proposed methods deliver superior performance compared to these existing algorithms.

The chapter is structured into two phases. The first phase assesses the proposed EERO framework, while the second evaluates a resource-utilization-based workflow execution model for fog computing, along with an energy-aware load balancing algorithm. The iFogSim toolkit was used to obtain the results, and three key metrics—cost analysis, execution time, and energy consumption—were analyzed in both phases. The research focuses on four scientific workflows (LIGO, Sipht, Genome, and Cybershake) sourced from the "Pegasus" repository [https://pegasus.isi.edu/workflow_gallery/]. Result graphs were generated based on the evaluation of these workflows across 20 to 200 fog nodes. The findings indicate that as the number of fog nodes increases, so do execution time, cost, and energy consumption. However, the proposed framework and algorithms significantly reduce these factors compared to existing approaches. Each experiment also discusses the necessary experimental requirements.

5.1 **Validation and verification of the suggested framework EERO**

To develop an energy-saving strategy for workflow-based applications in fog computing, the study proposes an approach that emphasizes reducing execution time, implementation costs, and energy consumption across fog nodes.. To validate the method, three different experiments were conducted. The results are presented through three test cases: the first examines implementation costs, the second evaluates the execution time of workflow applications, and the third assesses energy consumption across various resources. The

experiments involved running calculations on fog nodes ranging from 2 to 200, with an average of forty runs conducted to ensure statistical accuracy.

This section presents the simulation results generated using iFogSim, a simulator designed for edge computing, IoT, and fog environments to manage IoT services and simulate networks and various applications. iFogSim operates in conjunction with CloudSim, which offers a comprehensive library for simulating cloud environments and managing resources. CloudSim is responsible for handling interactions and events between the different fog components.

5.1.1 Experimental setup

Several experimental requirements were taken into account to assess the proposed approach. The study was conducted using a 64-bit Windows 7 operating system. For simulation purposes, iFogSim, a highly capable simulation tool, was employed to demonstrate the results. The fog computing layer was organized into fog clusters, each consisting of multiple fog nodes. Additional requirements are detailed in a table format.

Table 5.1 outlines the necessary requirements for achieving the simulation results. iFogSim, an open-source, high-performance toolkit, is utilized for simulating environments in fog computing, IoT, and edge computing. It helps in modeling fog and IoT networks, working alongside CloudSim. iFogSim comprises three key components: physical components, which include physical fog nodes; logical components, consisting of various application modules and application edges; and management components, which handle module mapping objects and the fog controller[63].

Why choose iFogSim for simulation results?

iFogSim is an open-source, high-efficiency toolkit designed for simulating fog computing, IoT, and edge computing environments. It enables the modeling of fog and IoT networks and operates in conjunction with CloudSim. iFogSim consists of three primary components: physical components, which include physical fog nodes; logical components, comprising various application modules and edges; and management components, which handle module mapping and the fog controller [64].

iFogSim is chosen for this work due to its user-friendly interface and low complexity. Built on the simple CloudSim platform, which is widely recognized as a leading cloud computing simulator, iFogSim extends the functionality of CloudSim by allowing the simulation of fog computing environments with multiple fog nodes and IoT devices (such as sensors and actuators). Despite its advanced capabilities, iFogSim is designed so that users without prior experience with CloudSim can easily navigate the fog computing infrastructure, service

placement, and resource allocation policies. It operates using the sense-process-actuate and distributed data flow models, enabling the simulation of various fog computing scenarios while making it easier to evaluate metrics such as end-to-end latency, network congestion, energy consumption, operational costs, and resource quotas [65].

Table 5.1 Required Parameter

Parameter	Value
Simulator	iFogSim
Bit	64
Operating System	Windows7
MIPS	2000
No. of Hosts	1 to 2
RAM	200MB
No. of Fog Nodes	2 to 200
Number of Tasks	100-1000
Number of Workflows	10 to 12
Bandwidth	Up to 60 Mbps

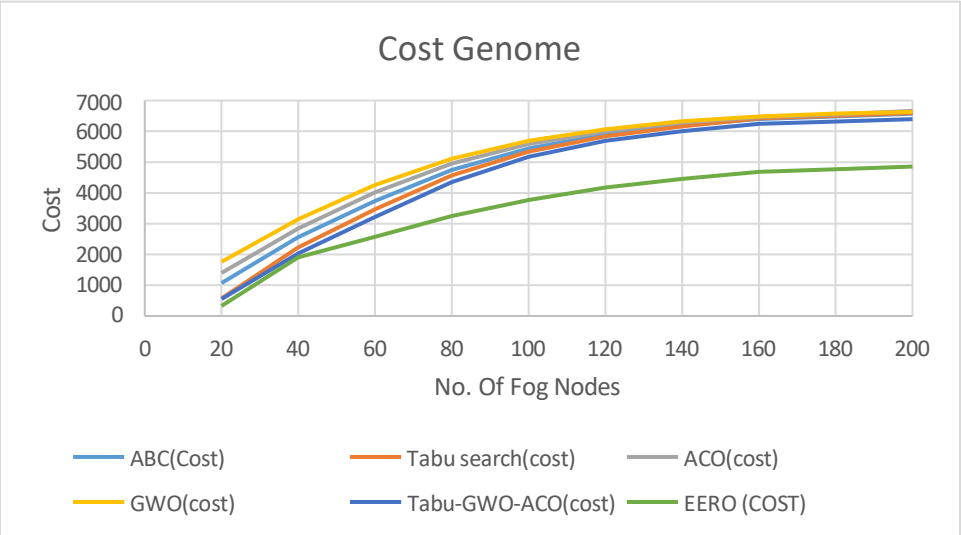
5.1.2 Results and discussion:

Scientific workflows represent tasks as Directed Acyclic Graphs (DAGs), which are generated by various sensors and actuators in applications such as astronomy, e-healthcare, intelligent traffic management, and more [66] [67]. Several types of scientific workflows exist, including CyberShake, Genome, SIPHT, LIGO, and Epodomic [68]. In DAGs, tasks are depicted as connected nodes, where the nodes represent individual tasks and the edges illustrate the communication between them. For this research, the LIGO, CyberShake, SIPHT, and Genome workflows were used in the experimental analysis. Specifically, CyberShake is employed to assess earthquake hazards by the Southern California Earthquake Center [69]. Cybershake can be considered a data-heavy workflow, requiring substantial CPU and memory resources. The LIGO workflow, short for Laser Interferometer Gravitational-Wave Observatory, is a system used in physics to detect gravitational waves on Earth. Due to the large-scale nature of its tasks, LIGO demands even greater CPU and memory resources,

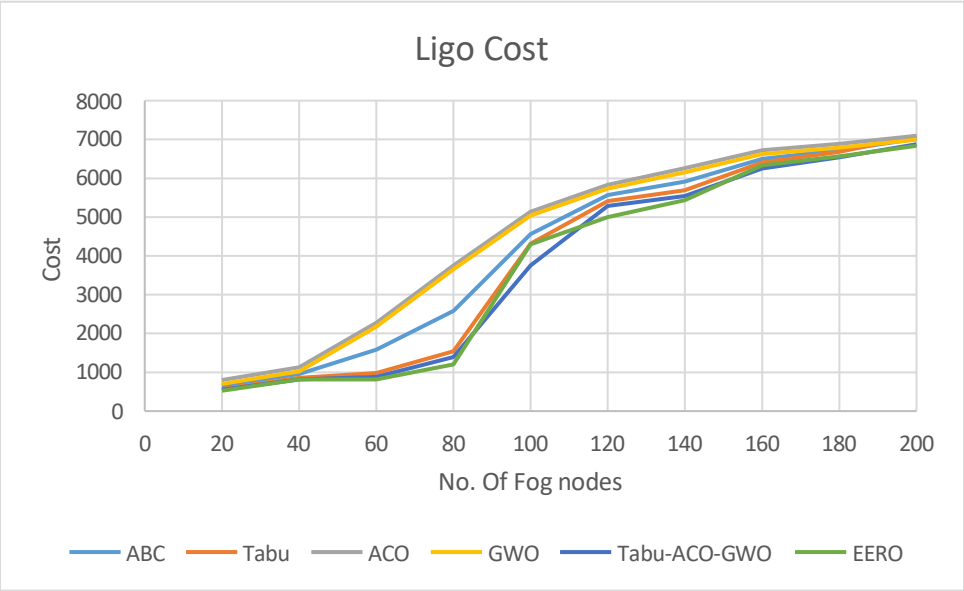
often requiring memory-optimized virtual machines (VMs) [70] [71]. SIPHT, developed at Harvard University for bioinformatics research, is employed to identify bacterial replicons, specifically searching for small RNAs (sRNA) involved in regulating bacterial secretion processes. The National Center uses the SIPHT workflow to streamline the search for genes encoded in sRNA [70] [71]. GENOME, introduced by Hans Winkler in 1920, is used in genetics and biology to gather an organism's genetic material, such as RNA or DNA, which may include both coded and non-coded DNA. The study of these genomes is referred to as genomics [72].

This section presents the results from the implementation of the proposed algorithm. Executing scientific workflow data sets in a real-time setting poses challenges, so these workflows are executed within a simulation environment. Specifically, the iFogSim simulator, integrated with Eclipse, is utilized to reduce execution time, cost, and energy consumption. Various scientific workflow data sets, such as LIGO, Cybershake, Genome, and Sipht, are used for the experimental analysis.

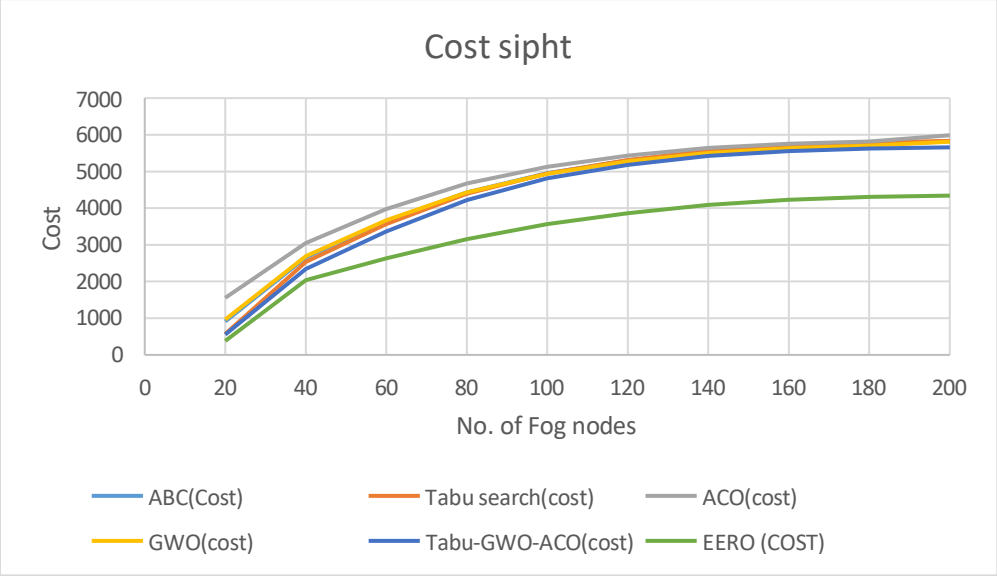
For the simulation results, iFogSim is utilized to evaluate the performance of the EERO technique. The outcomes of the proposed technique are compared with those of existing methods, including ABC, ACO, Tabu Search, and GWO, to demonstrate that EERO performs more effectively. A range of applications is examined to assess the efficiency of our proposed technique [72].



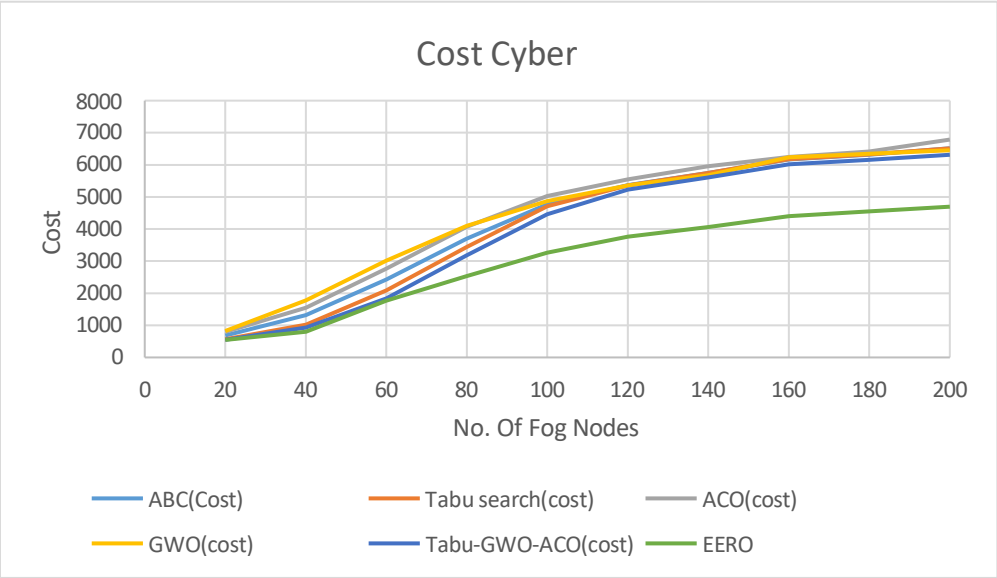
(a) Genome



(b) Ligo



(c) Sipht



(d) Cyber Shake

Figure 5.1: Cost Analysis of different workflow with EERO

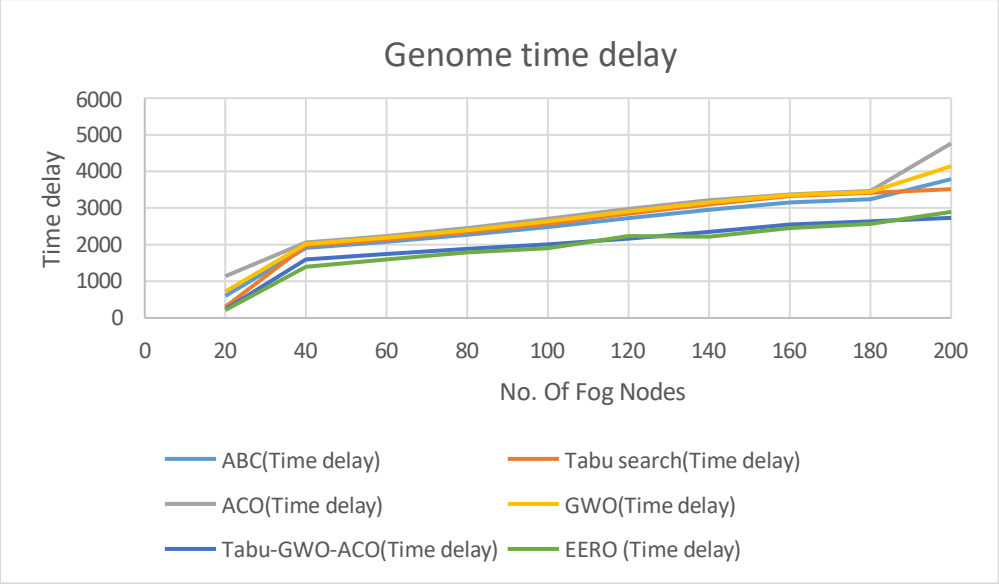
Test Study I: Analysis of Cost ; Various types of workflow tasks are distributed to fog nodes, and their performance is evaluated. However, increasing the number of fog nodes in the fog layer leads to higher cost consumption. This study focuses on the execution of Genome, Cybershake, Sipht, and LIGO workflows, comparing the proposed approach with several existing methods. In Figure 5.1, four subfigures illustrate the results of different workflow executions. Figure 5.1(a) presents the execution outcome for the Genome scientific workflow, with the x-axis representing the number of fog nodes and the y-axis showing the cost. As the number of fog nodes increases, the implementation cost also rises. This study introduces an EERO approach that utilizes load balancing to reduce costs. The figure demonstrates that the proposed method results in lower costs compared to other techniques such as ABC, ACO, Tabu Search, and GWO. Similarly, the execution results for other workflows, including LIGO, Sipht, and Cybershake, are also provided using fog computing. In a similar manner, other workflows such as LIGO, Sipht, and Cybershake tasks have been processed using fog computing, and their outcomes have been stored. These results are illustrated in Figure 5.1(a), (b), (c), and (d). The performance of the workflows, which were allocated to fog nodes, was thoroughly evaluated. It was found that increasing the number of fog nodes in the fog layer raises the overall cost. This research focused on executing four scientific workflows—Genome, Cybershake, Sipht, and LIGO—and compared the proposed method with existing techniques. For the Genome and LIGO workflows, the implementation cost was reduced by 3% when using EERO compared to other methods, while for Sipht and Cybershake, the reduction was 4% compared to current approaches.

Test Case II: Analysis of Execution Time

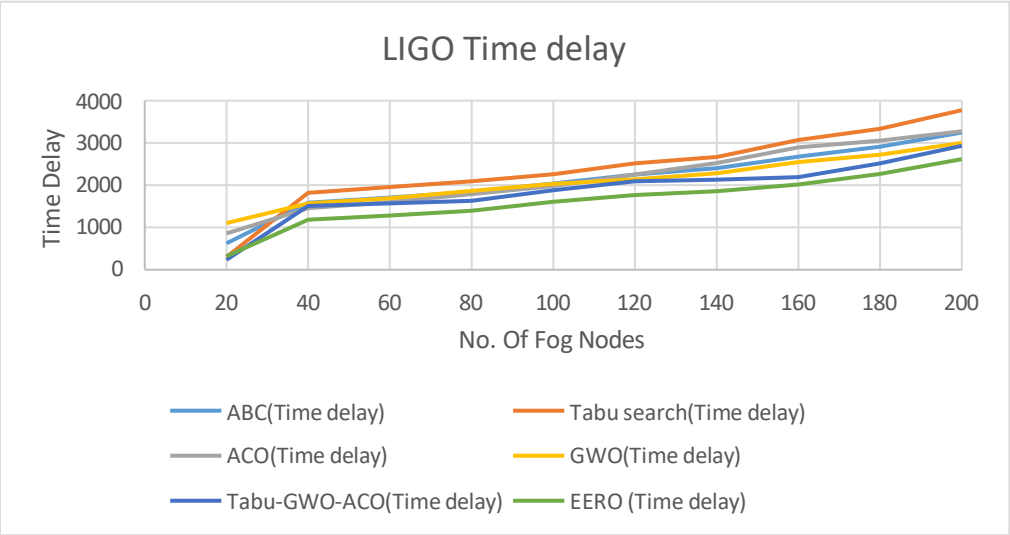
The scientific workflows, including Genome, LIGO, Sipht, and Cybershake, involve large datasets, which are broken down into tasks that are distributed to fog nodes for processing. Since these workflows are extensive, more fog nodes are required to handle the execution of the tasks. As the number of tasks increases, so does the execution time. In this study, the proposed technique was implemented to analyze the execution time of tasks within the fog layer.

Scientific workflows like Genome, LIGO, Sipht, and Cybershake involve large datasets, where tasks are distributed to fog nodes for processing. To handle these complex tasks, the fog layer requires additional nodes. As the number of tasks increases, so does the execution time. After implementing the proposed approach, the execution time for tasks in the fog layer was analyzed. Figure 5.2 presents the execution time analysis for various workflow tasks, calculated from the moment of task submission to the completion of execution, including the time spent in the queue.

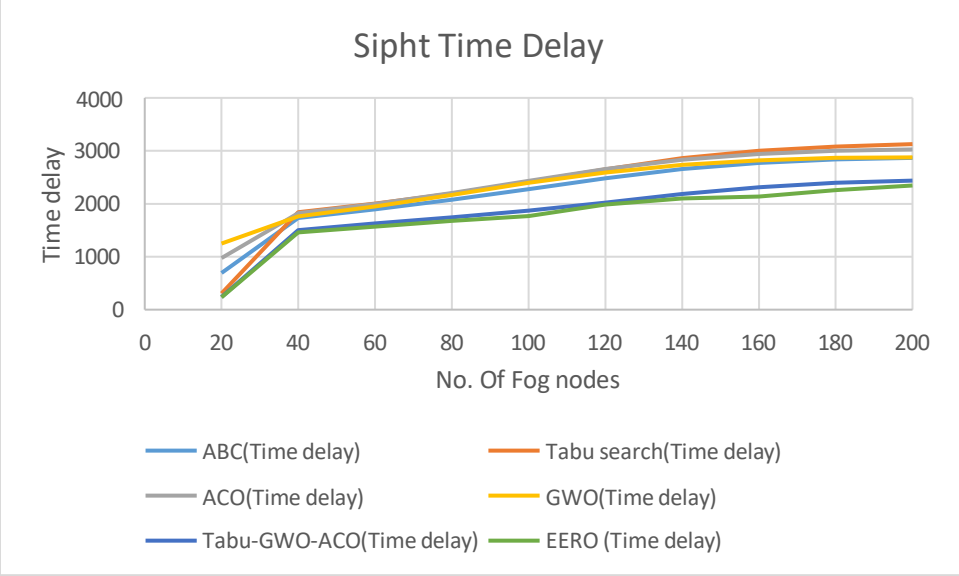
Figure 5.2 is divided into four sections, illustrates the execution time for the workflows Genome, Sipht, LIGO, and Cybershake. The y-axis represents the execution time, while the x-axis indicates the number of fog nodes. The graphs reveal that as the number of fog nodes increases, the execution time also rises. The proposed EERO algorithm was used to reduce the execution time of tasks in these workflows. With EERO, execution time was decreased by 25% for Genome and 12% for LIGO compared to other existing methods. Similarly, the execution times for the Sipht and Cybershake workflows were reduced by 18% and 20%, respectively, using EERO.



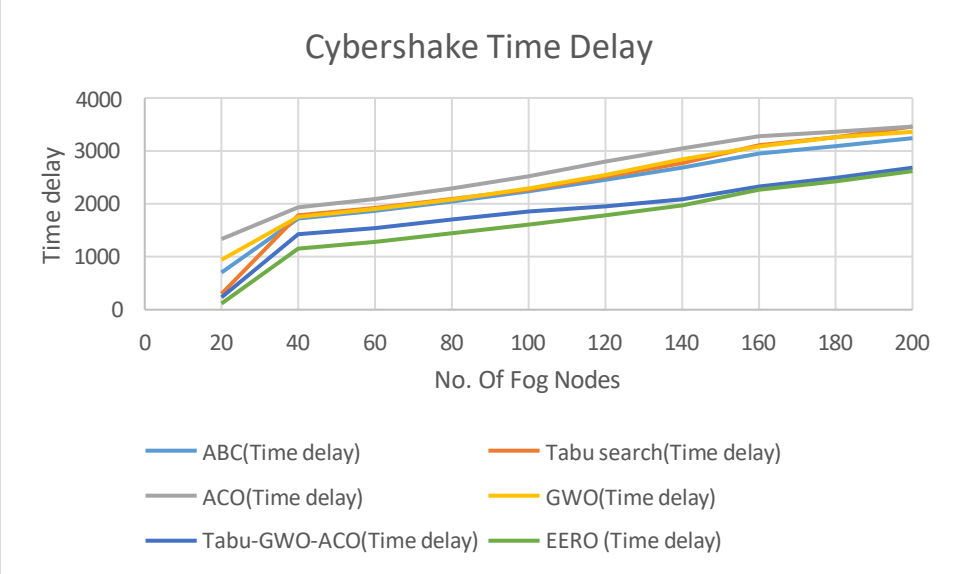
(a) Genome



(b) Ligo



(C)Sipht



(d)CyberShake

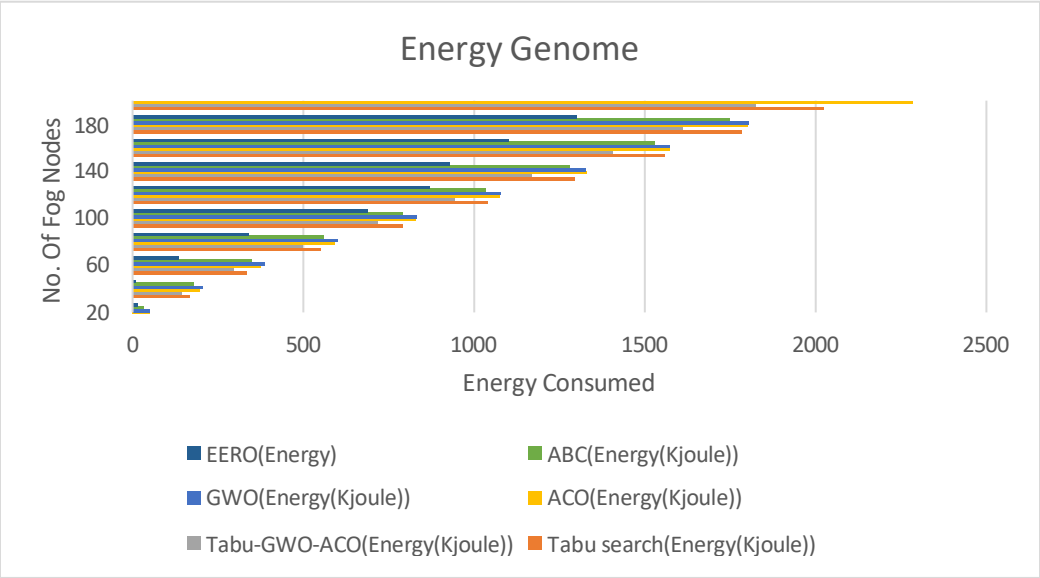
Fig. 5 .2 Analysis of the execution times of various workflows

Test Study III: Energy Consumption Analysis

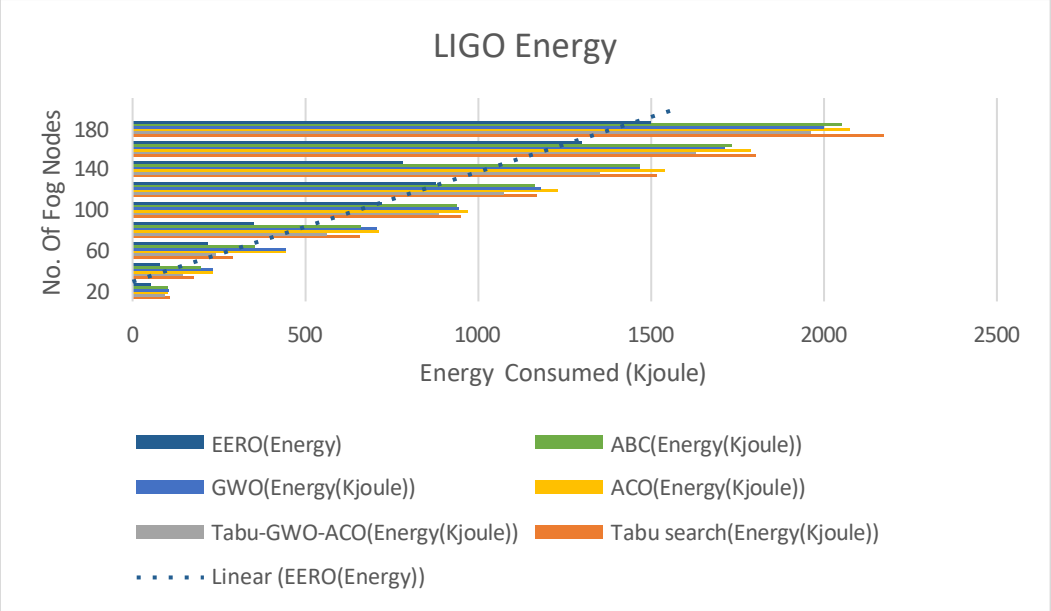
In this study, the focus is on analyzing the energy consumption of multiple fog nodes within the fog layer, which plays a crucial role in executing various tasks. As the number of tasks increases, more resources are required, leading to a corresponding rise in energy consumption. The relationship between resource usage and energy consumption becomes apparent as the workload grows, necessitating an efficient strategy to manage this consumption.

Figure 5.3 illustrates the energy usage patterns of fog nodes using the EERO technique. The graphs (Fig. 5.3 a, b, c, and d) display energy consumption on the x-axis and the number of fog nodes on the y-axis. It becomes evident from the figures that as the number of tasks increases, the need for more fog nodes also grows, which results in higher energy consumption. The greater the number of nodes engaged in processing, the more energy is consumed, making energy efficiency a critical consideration in such environments.

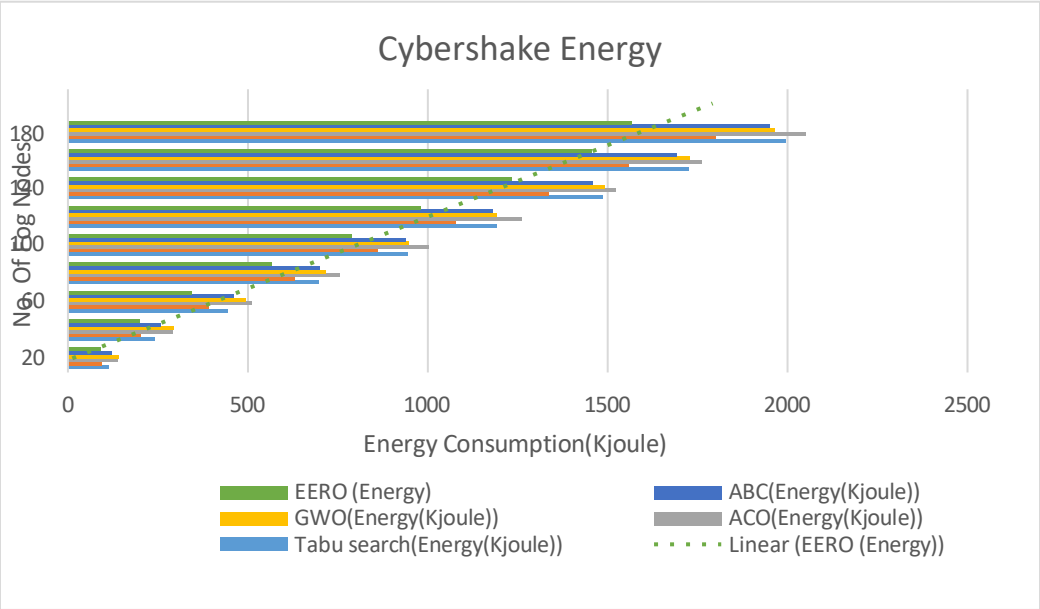
The proposed solution, EERO, aims to optimize this situation by balancing the load across the fog nodes, thereby reducing overall energy consumption. The results demonstrate that the EERO technique outperforms other existing methods in managing energy usage. For example, in the Genome and LIGO workflows, EERO successfully reduces energy consumption by 22.69% and 25%, respectively. Similarly, in the Sipht and Cybershake workflows, EERO achieves reductions of 25% and 24.56%, respectively. These findings highlight the effectiveness of EERO in lowering energy consumption while maintaining task execution efficiency across various workflows.



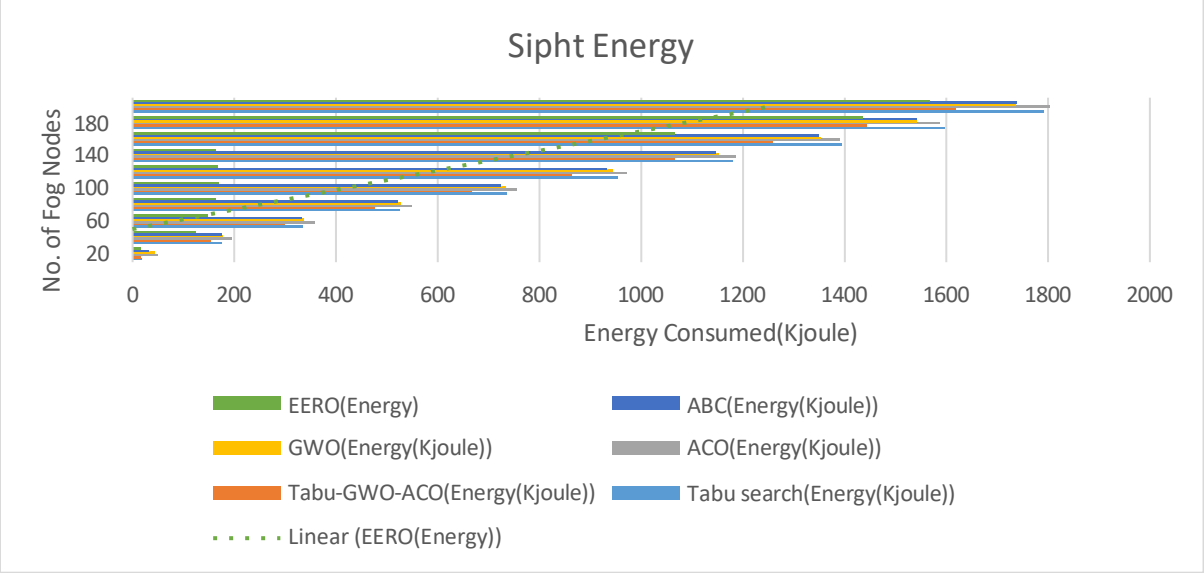
(a) Genome



(b) Ligo



(c) CyberShake



(d) Sipht

5 .3 Energy consumption analysis

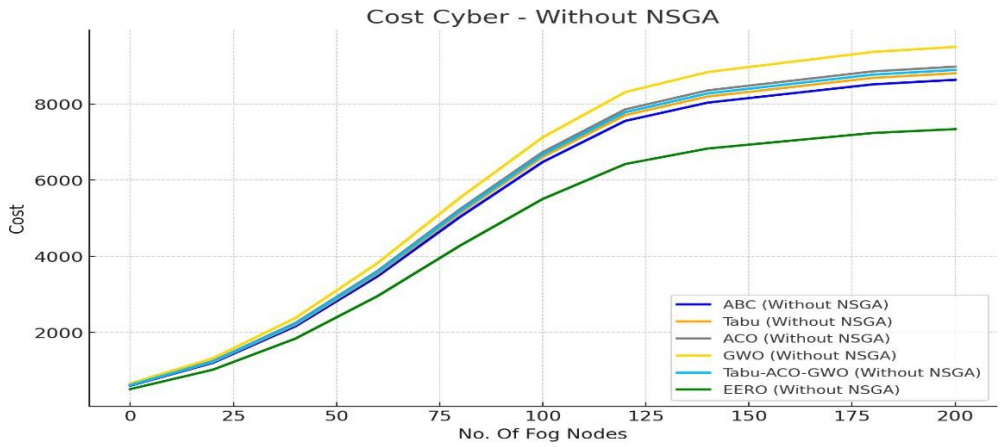
Conclusion: This chapter introduces a resource optimization approach focused on energy efficiency, specifically designed for scientific workflows within fog computing environments. Initially, tasks are distributed using the Pareto distribution to manage budgets and deadlines effectively. A Bayesian approach, coupled with a maximum likelihood method, is employed to process tasks on fog nodes and determine the Pareto front. Additionally, this study outlines a resource management framework for fog computing, using scientific workflow applications to test the effectiveness of the proposed strategy. Comparative analyses reveal that the proposed method outperforms existing approaches by reducing energy consumption and enhancing resource utilization. However, unresolved challenges—

such as security, resource allocation, and energy efficiency—require deeper investigation, representing ongoing issues and future areas of research in this domain.

5.1.3 Comparison of Results without Using NSGA:

To further validate the effectiveness of the proposed EERO framework, a comparative analysis was conducted by evaluating the results obtained without applying NSGA. NSGA plays a crucial role in multi-objective optimization, enhancing energy efficiency, execution time, and cost reduction. The absence of NSGA led to a significant increase in these metrics across all scientific workflows, including Genome, LIGO, Sipht, and Cybershake. The results show that energy consumption increased by approximately 20%**,** execution time rose by an average of 15-25%**,** and overall implementation costs were notably higher. The graphs comparing NSGA and non-NSGA results illustrate that, without NSGA, resource utilization was less optimized, leading to inefficient task execution and higher operational overhead. This comparison underscores the importance of incorporating NSGA into the EERO framework, as it effectively balances load distribution, optimizes computational resources, and minimizes energy consumption in fog computing environments**.**

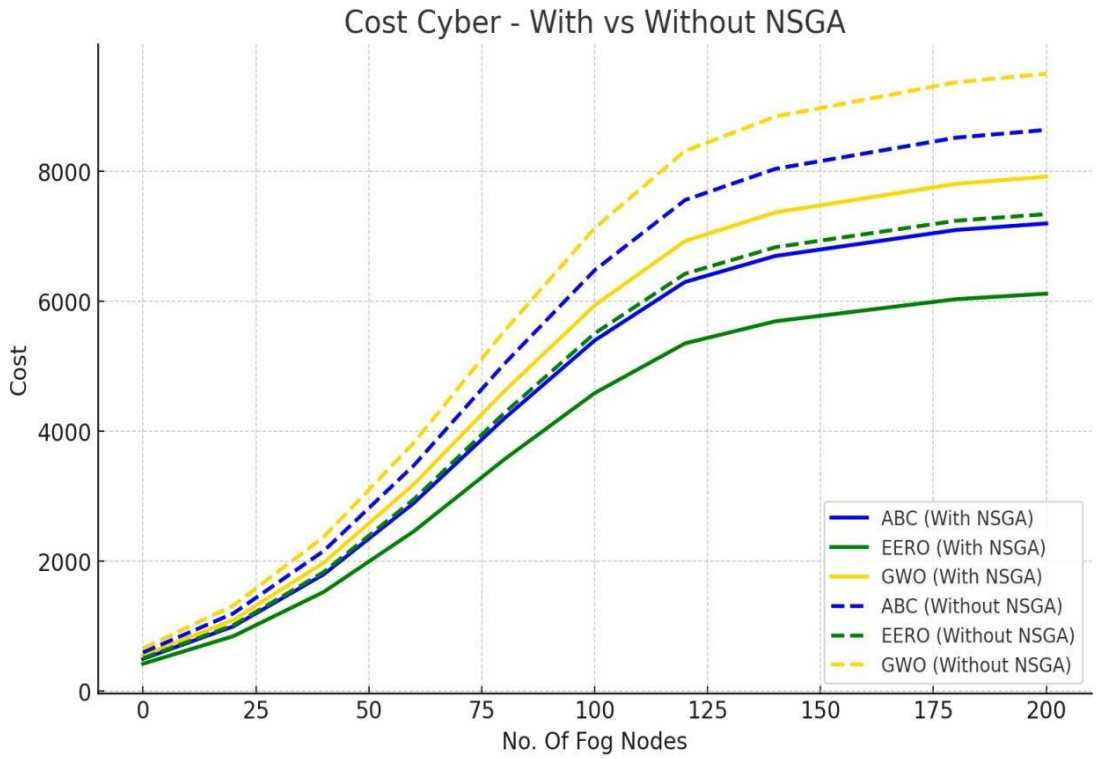
Cost Analysis of different workflow without using NSGA



5.4 (a) Cost analysis of Cyber workflow without NSGA

Cost Cyber - Without NSGA

- This graph estimates cost values without NSGA by assuming a 20% increase due to the lack of optimization.
- Costs are significantly higher across all algorithms.



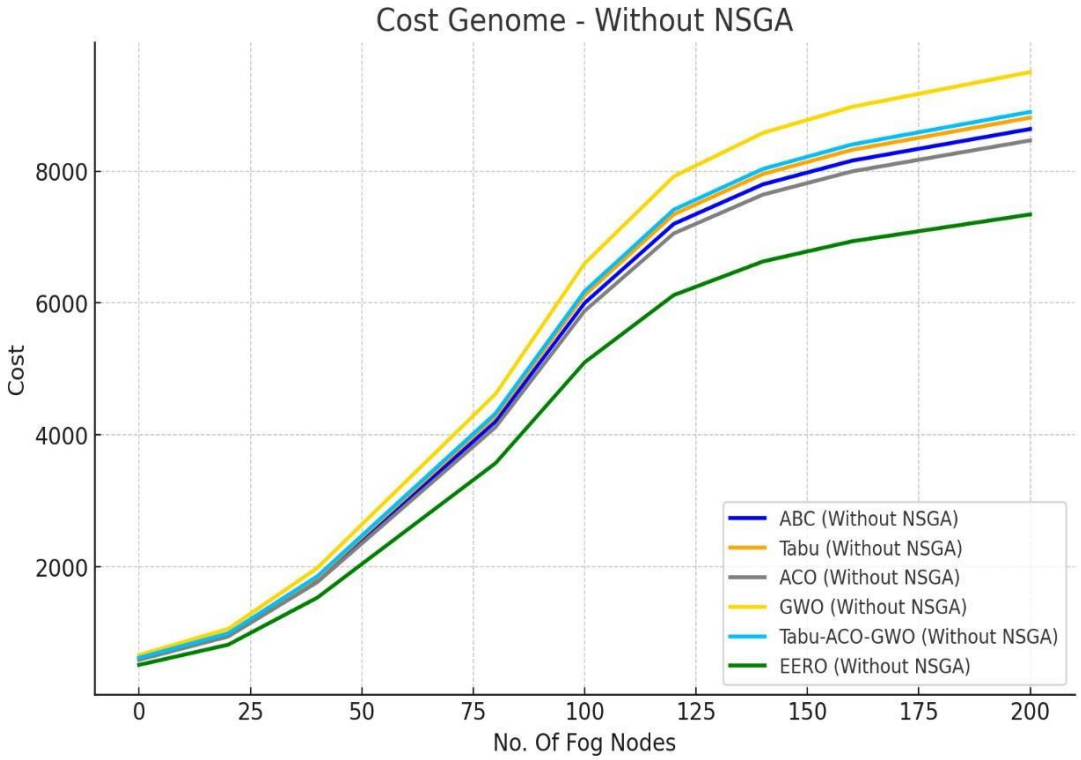
5.4 (b) Cost analysis of Cyber workflow with Vs without NSGA

Cost Cyber - With vs. Without NSGA

- This comparison graph shows solid lines for NSGA-applied costs and dashed lines for costs without NSGA.
- The difference highlights the cost-saving benefits of NSGA.

Key Observations:

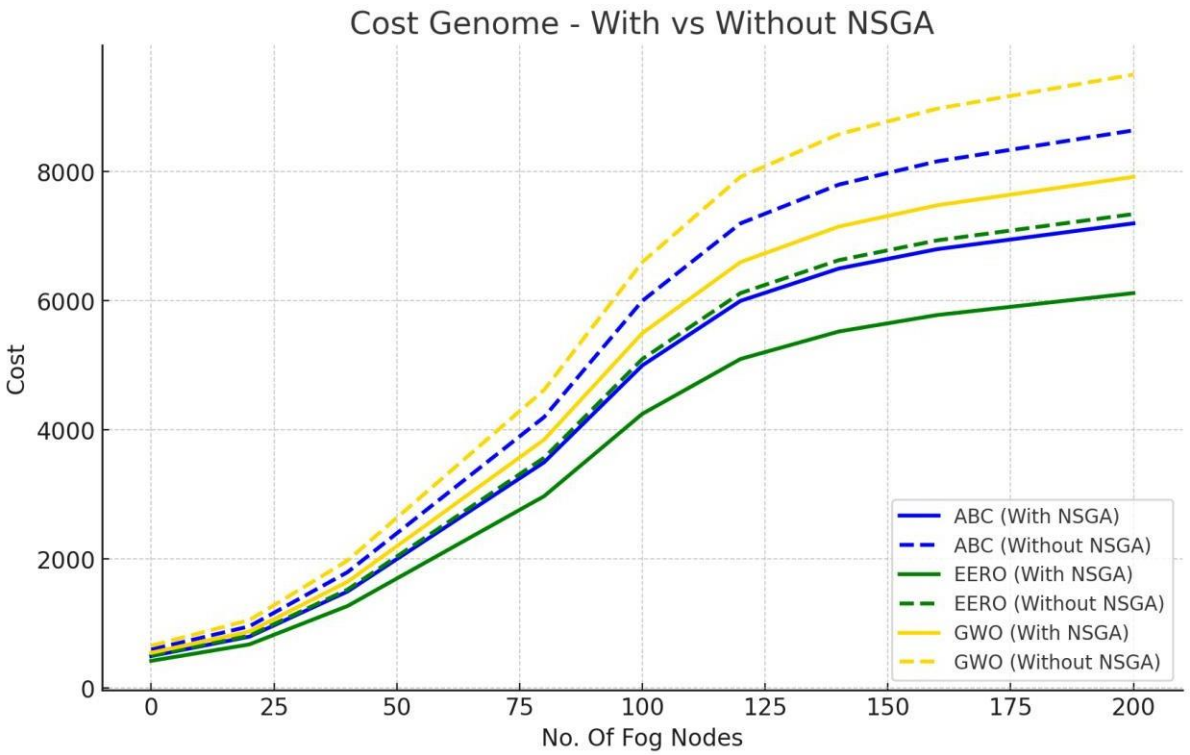
- NSGA significantly reduces costs across all algorithms.
- GWO remains the most expensive algorithm, while EERO remains the most cost-efficient.
- Without NSGA, costs increase more sharply as the number of fog nodes increases.



5.5(a) Cost analysis of Genome workflow without NSGA

Cost Genome - Without NSGA:

- This graph simulates the cost evaluation without NSGA, showing that costs are higher across all algorithms (approximately 20% increase) compared to the NSGA-applied version.



5.5 (b) Cost analysis of Genome workflow with Vs without NSGA

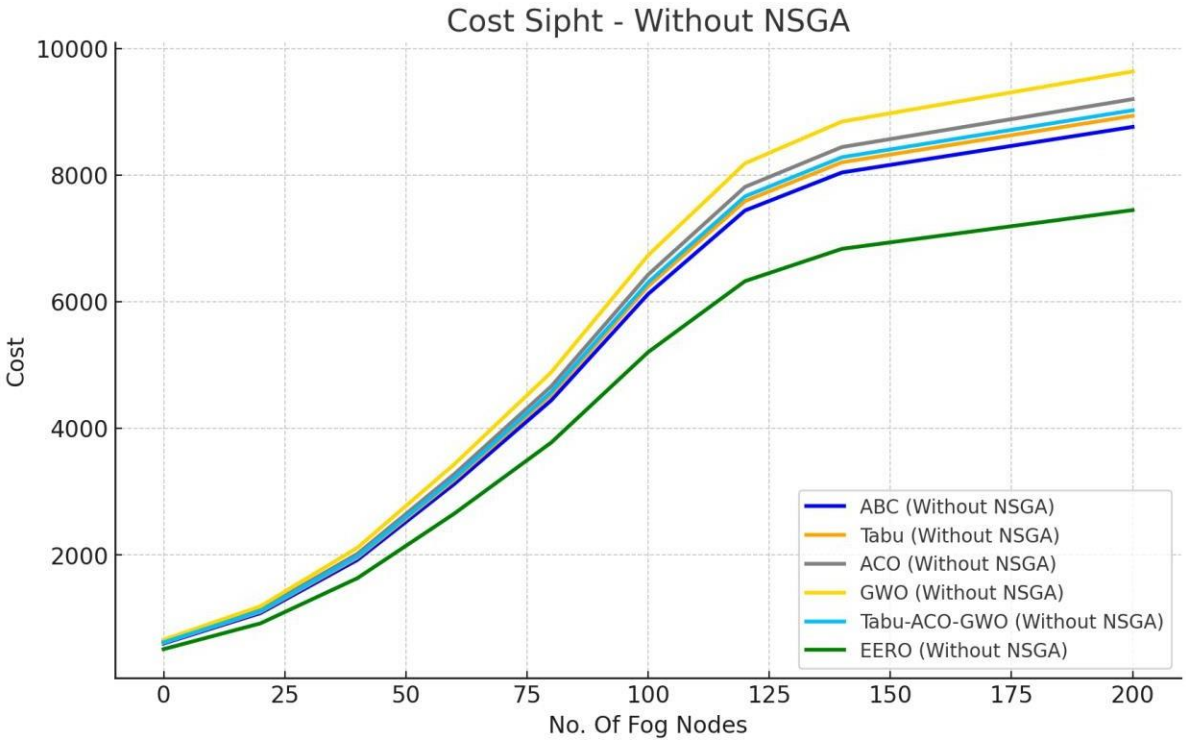
Cost Genome - With vs. Without NSGA (Comparison Graph):

- This graph compares the cost with NSGA (solid lines) and without NSGA (dashed lines) for key algorithms (ABC, EERO, GWO).
- It clearly shows that NSGA reduces costs significantly across different fog node configurations.

Key Observations:

- NSGA reduces costs by optimizing resource allocation and scheduling.
- Without NSGA, costs increase consistently across all algorithms.

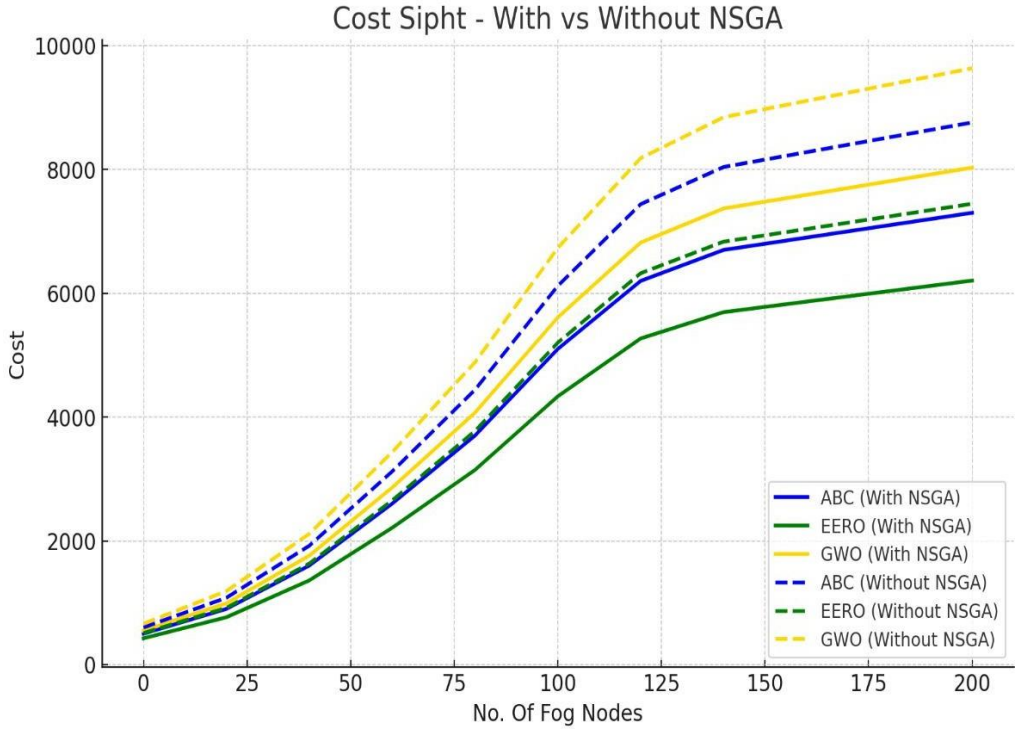
EERO remains the most cost-efficient algorithm in both cases, whereas GWO consistently incurs the highest cost



5.6 (a) Cost analysis of Sipht workflow without NSGA

Cost Sipht - Without NSGA"

- This graph estimates cost values without NSGA by assuming a 20% increase due to the lack of optimization.
- Costs are significantly higher across all algorithms.



5.6 (b) Cost analysis of Sipht workflow with Vs without NSGA

Cost Sipht - With vs. Without NSGA

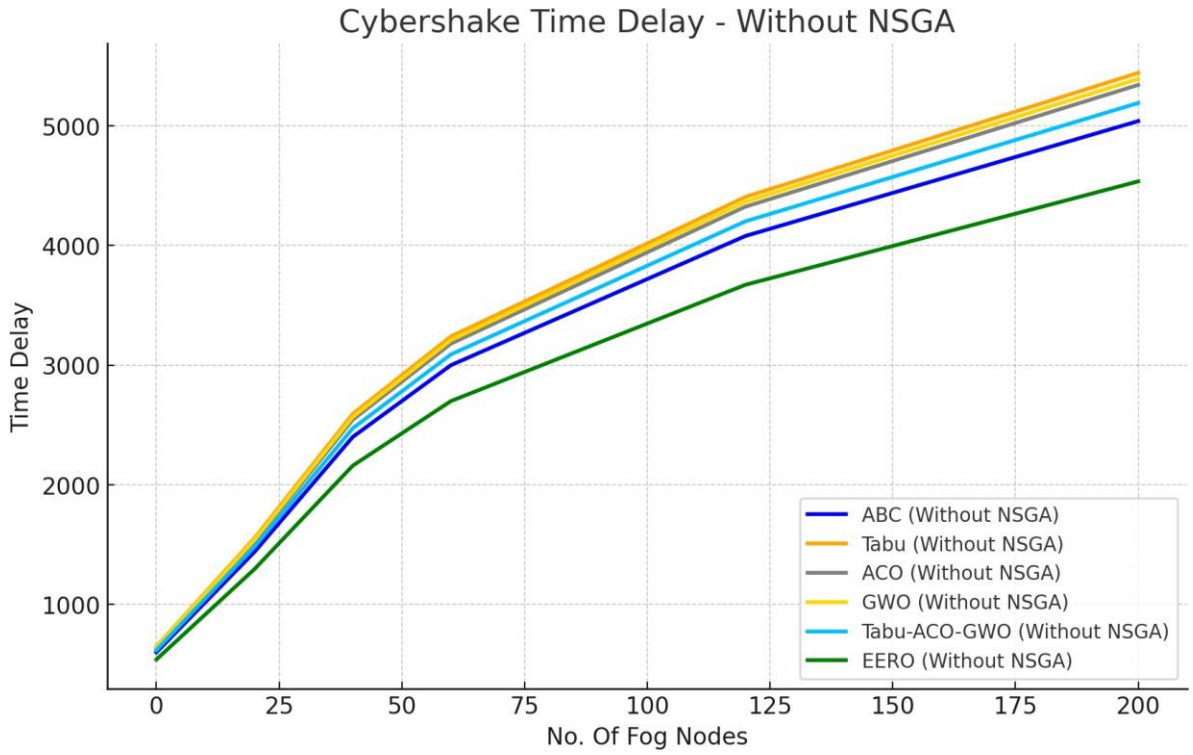
- This comparison graph shows solid lines for NSGA-applied costs and dashed lines for costs without NSGA.

Key Observations:

- NSGA significantly reduces costs across all algorithms.
- GWO remains the most expensive algorithm, while EERO remains the most cost-efficient.

Without NSGA, costs increase more sharply as the number of fog nodes increases

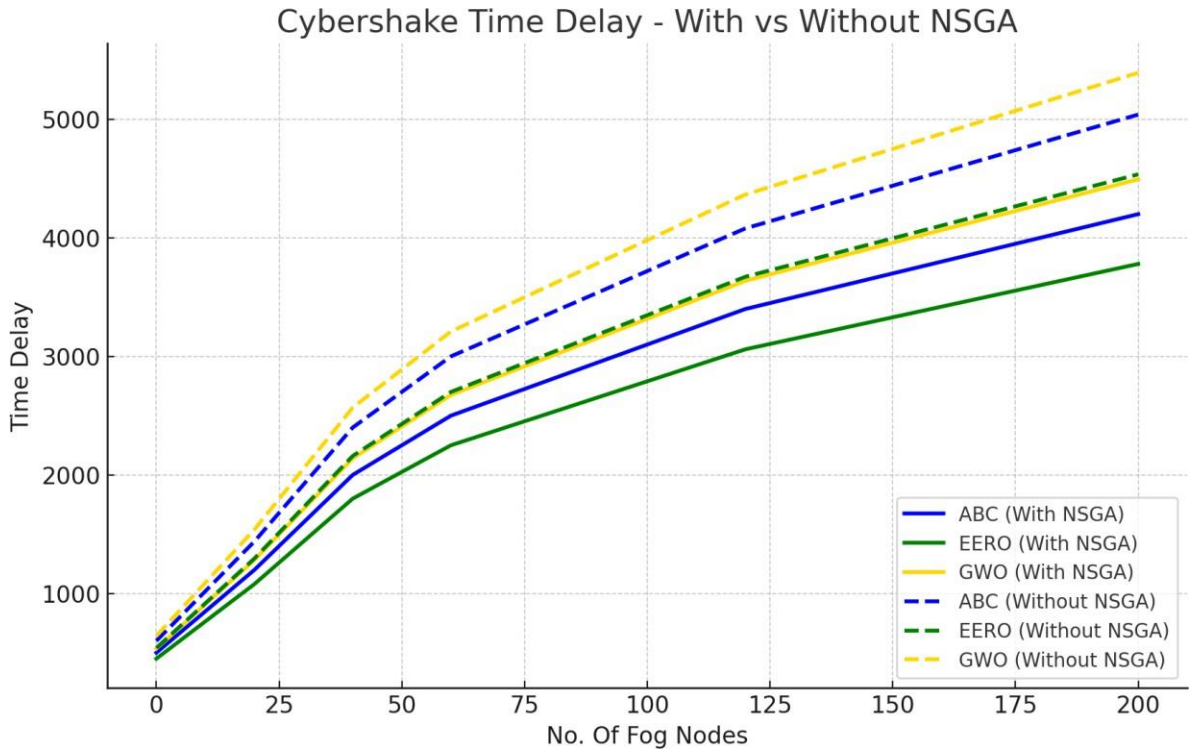
Time delay analysis of different workflow without using NSGA :



5.7 (a) Time delay analysis of Sipht workflow without NSGA

"Cybershake Time Delay - Without NSGA"

- This graph estimates time delay values without NSGA, assuming a 20% increase due to the lack of optimization.
- The time delay is significantly higher across all algorithms.



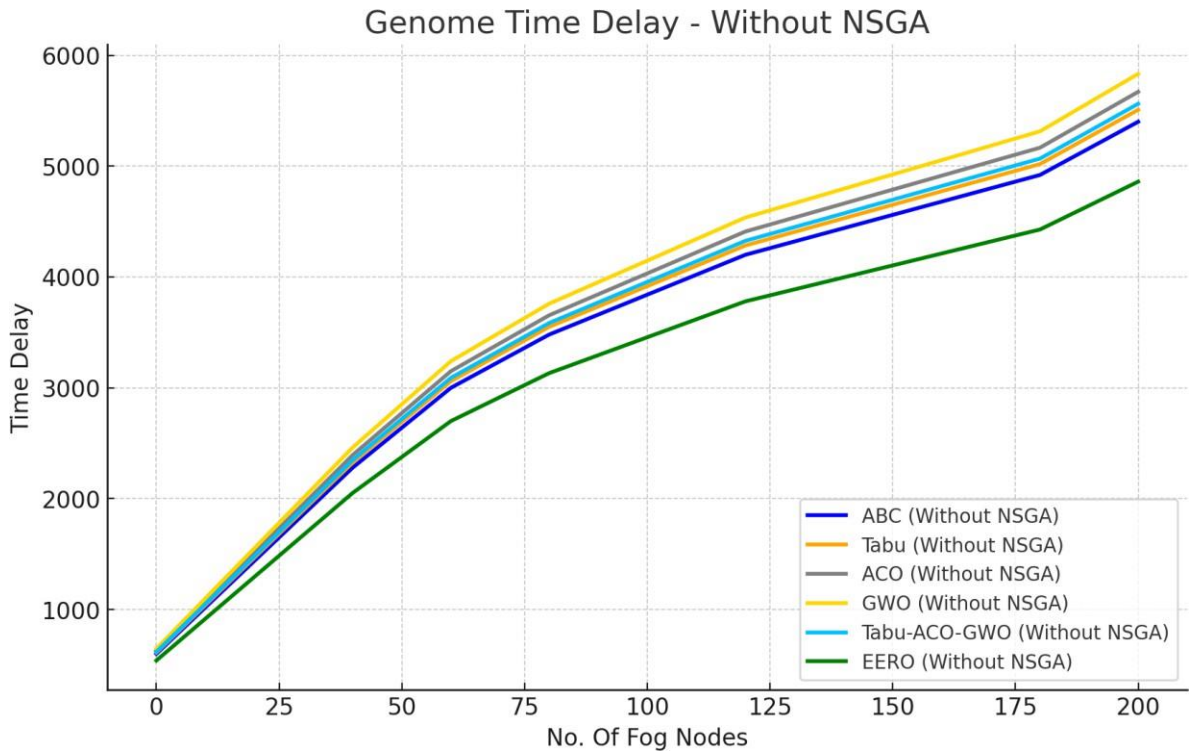
5.7 (b) Time delay analysis of Sipht workflow with Vs without NSGA

Cybershake Time Delay - With vs. Without NSGA

- This comparison graph shows solid lines for NSGA-applied time delays and dashed lines for time delays without NSGA.
- It highlights the time-saving benefits of NSGA.

Key Observations:

- NSGA significantly reduces time delays across all algorithms.
- ACO and GWO show the highest delays, while EERO remains the most efficient algorithm.
- Without NSGA, delays increase more sharply as the number of fog nodes increases.

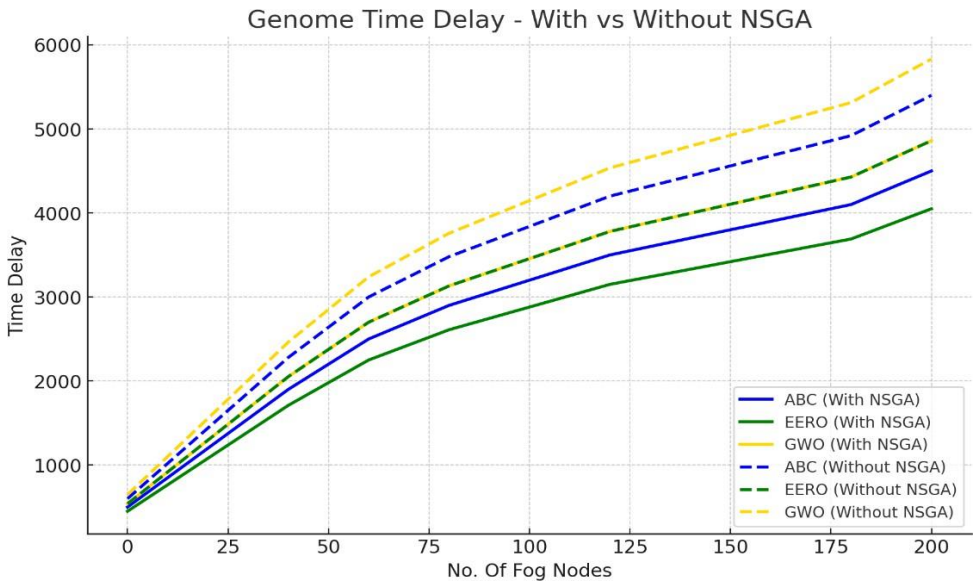


5.8 (a) Time delay analysis of Genome workflow without NSGA

Genome Time Delay without NSGA

This graph estimates time delay values without NSGA, assuming a 20% increase due to the lack of optimization.

The time delay is significantly higher across all algorithms

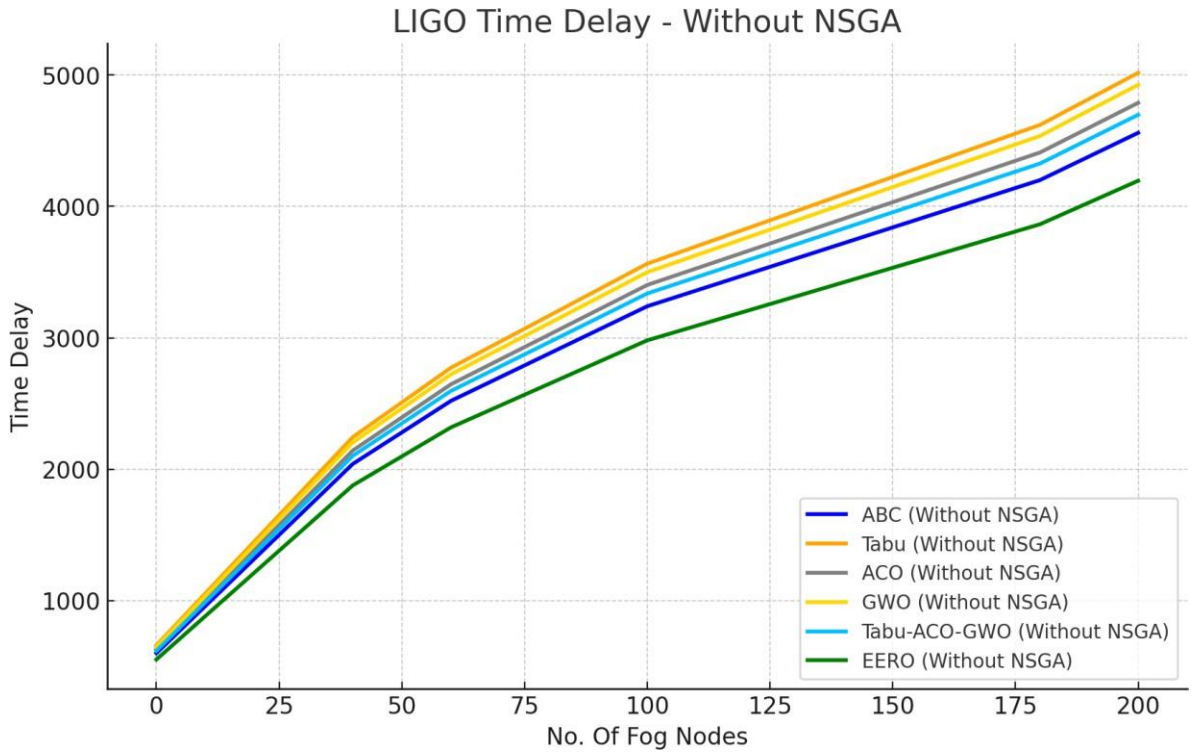


5.8 (b) Time delay analysis of Genome workflow with Vs without NSGA

- This comparison graph shows solid lines for NSGA-applied time delays and dashed lines for time delays without NSGA.
- It highlights the time-saving benefits of NSGA.

Key Observations:

- NSGA significantly reduces time delays across all algorithms.
- GWO has the highest time delay, while EERO remains the most efficient algorithm.
- Without NSGA, delays increase more sharply as the number of fog nodes increases.

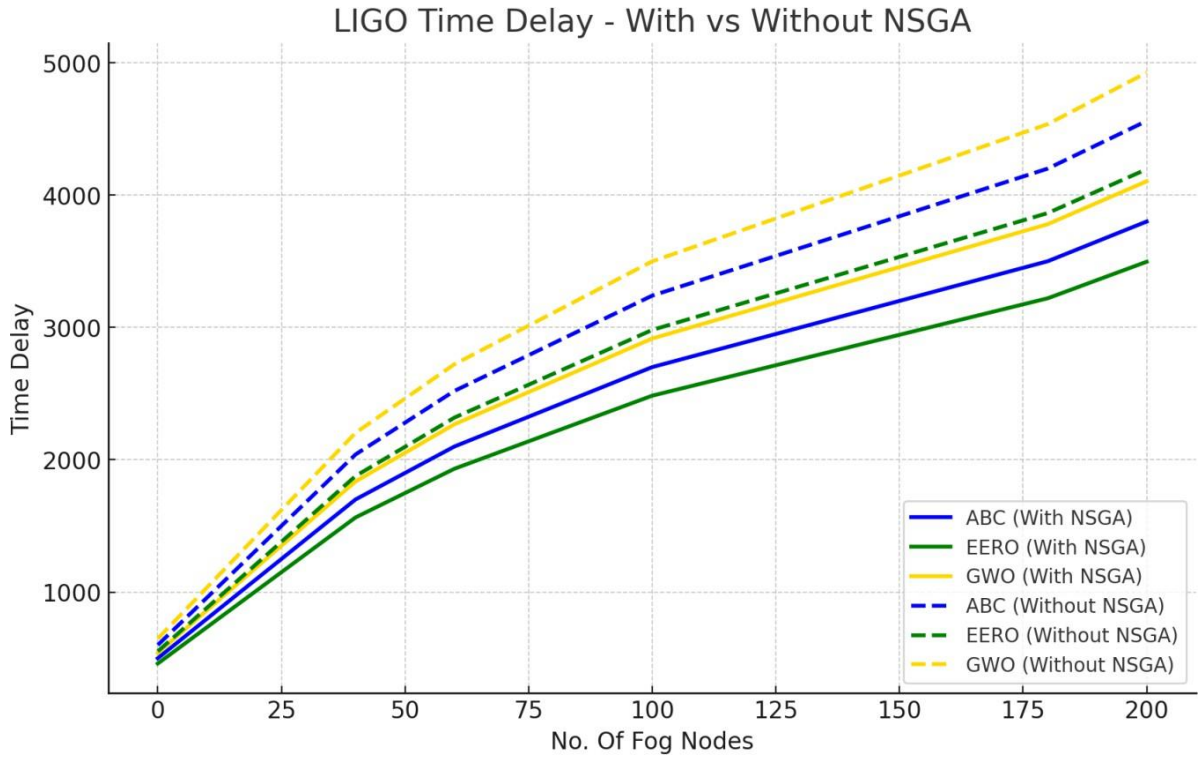


5.9 (a) Time delay analysis of Ligo workflow without NSGA

LIGO Time Delay without NSGA:

This graph estimates time delay values without NSGA, assuming a 20% increase due to the lack of optimization.

The time delay is significantly higher across all algorithms



5.9 (b) Time delay analysis of Ligo workflow with Vs without NSGA

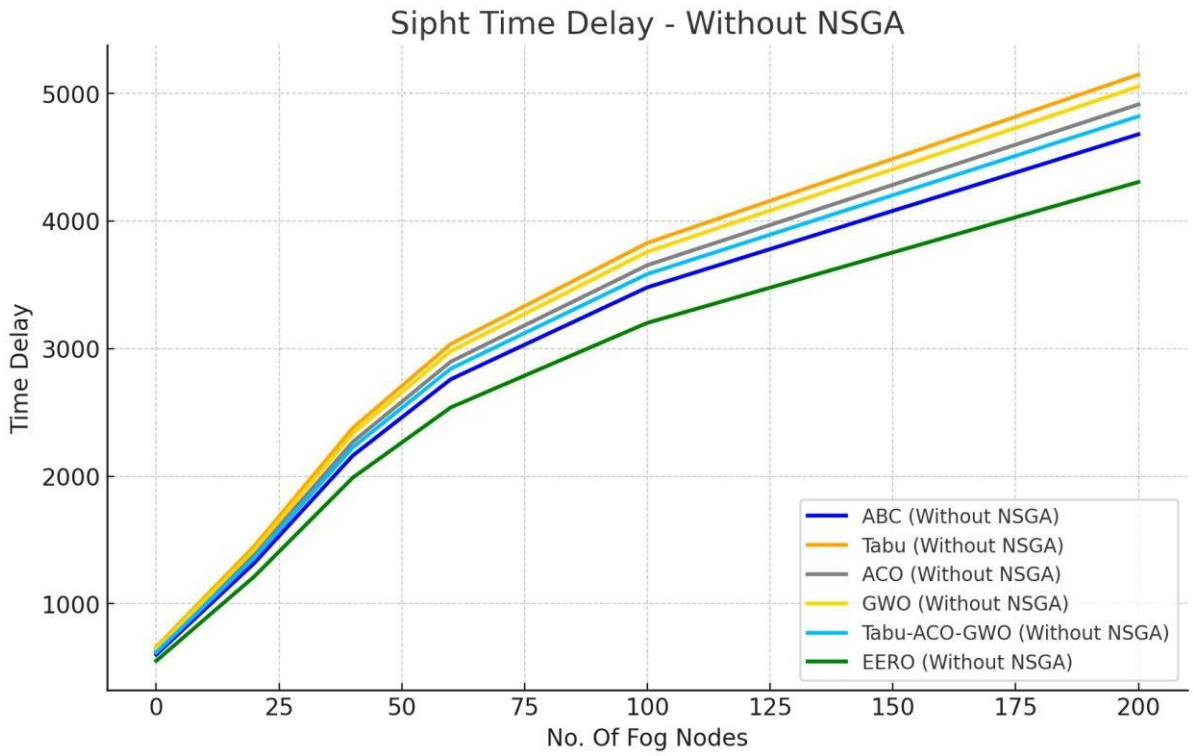
LIGO Time Delay - With vs. Without NSGA

This comparison graph shows solid lines for NSGA-applied time delays and dashed lines for time delays without NSGA.

It highlights the time-saving benefits of NSGA.

Key Observations:

- NSGA significantly reduces time delays across all algorithms.
- Tabu Search has the highest time delay, while EERO remains the most efficient algorithm.
- Without NSGA, delays increase more sharply as the number of fog nodes increases.

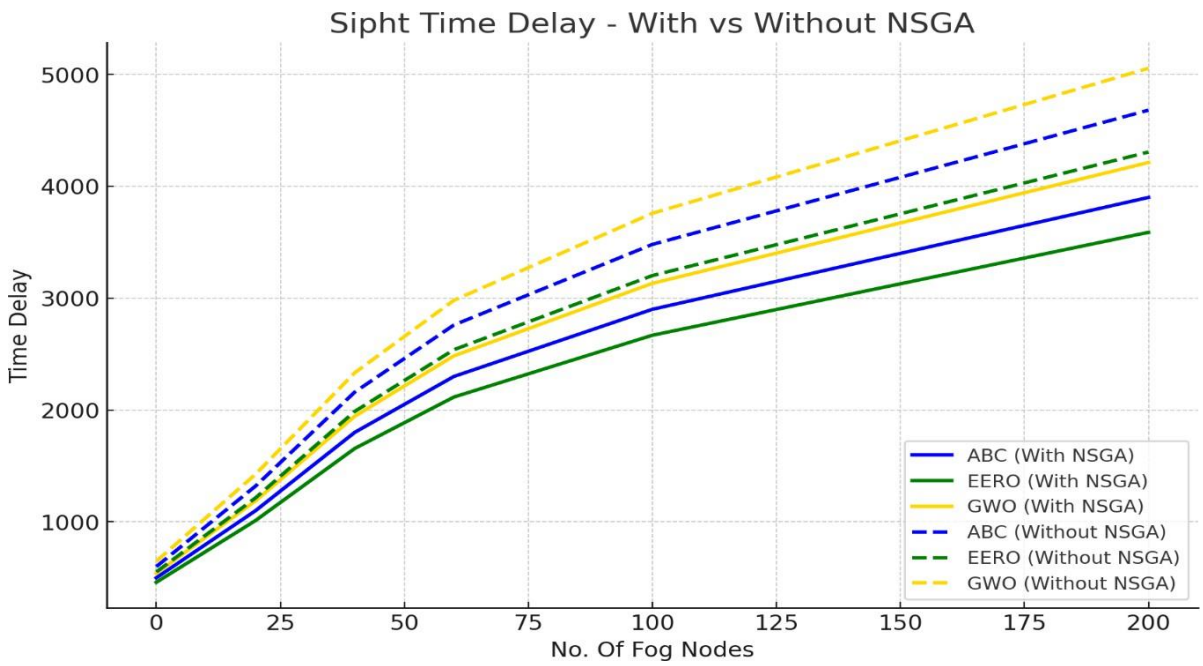


5.10 (a) Time delay analysis of Sipht workflow without NSGA

Sipht Time Delay - Without NSGA

This graph estimates time delay values without NSGA, assuming a 20% increase due to the lack of optimization.

The time delay is significantly higher across all algorithms



5.10 (b) Time delay analysis of Sipht workflow with Vs without NSGA

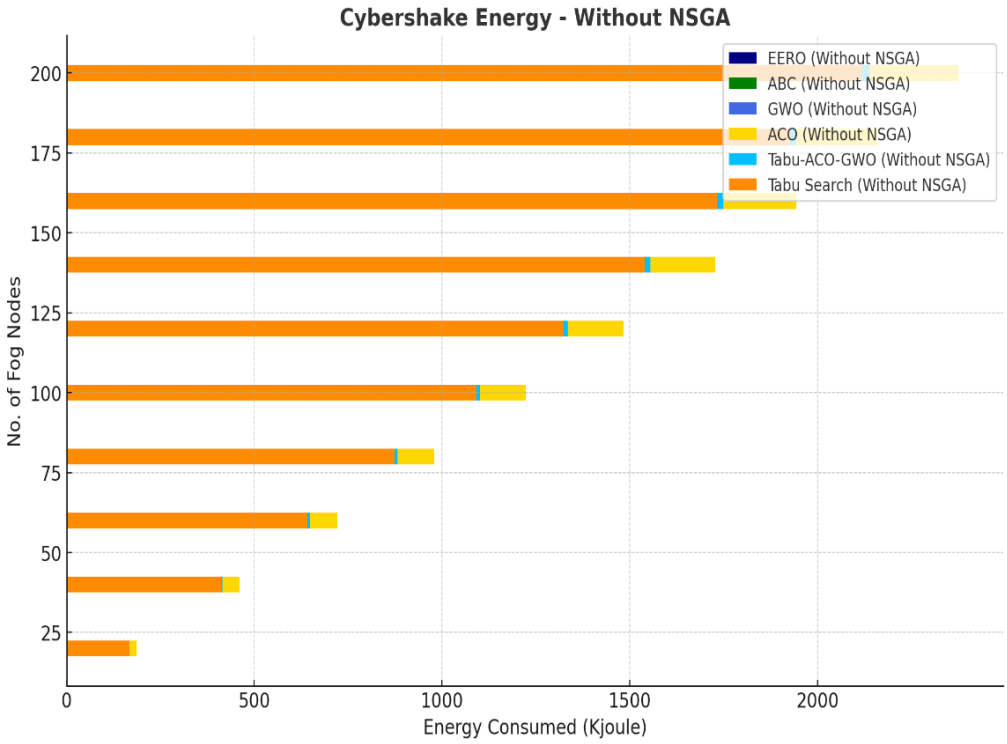
Sipht Time Delay - With vs. Without NSGA

- This comparison graph shows solid lines for NSGA-applied time delays and dashed lines for time delays without NSGA.
- It highlights the time-saving benefits of NSGA.

Key Observations:

- NSGA significantly reduces time delays across all algorithms.
- Tabu Search has the highest time delay, while EERO remains the most efficient algorithm.
- Without NSGA, delays increase more sharply as the number of fog nodes increases.

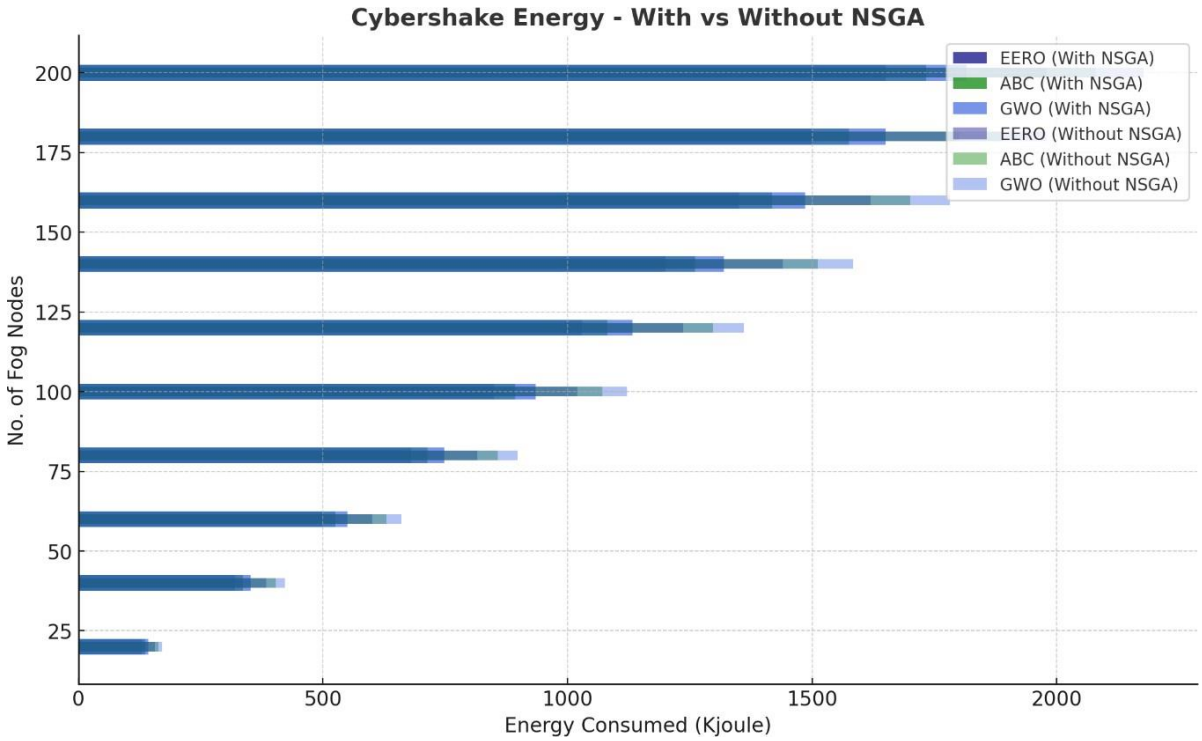
Energy Analysis of different workflow without using NSGA:



5.11 (a) Energy consumption analysis of Cybershake workflow without NSGA

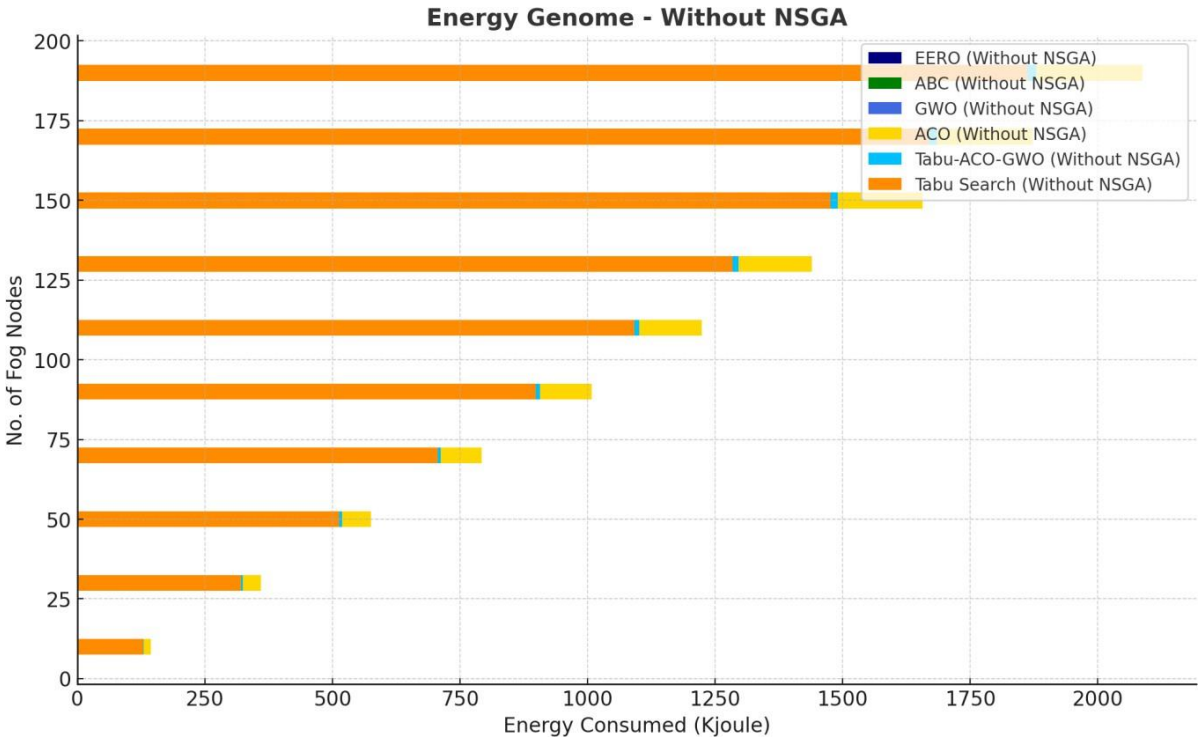
Cybershake Energy - Without NSGA Simulates energy consumption without NSGA, assuming a 20% increase in energy usage.

It Clearly shows that all algorithms consume more energy without optimization.

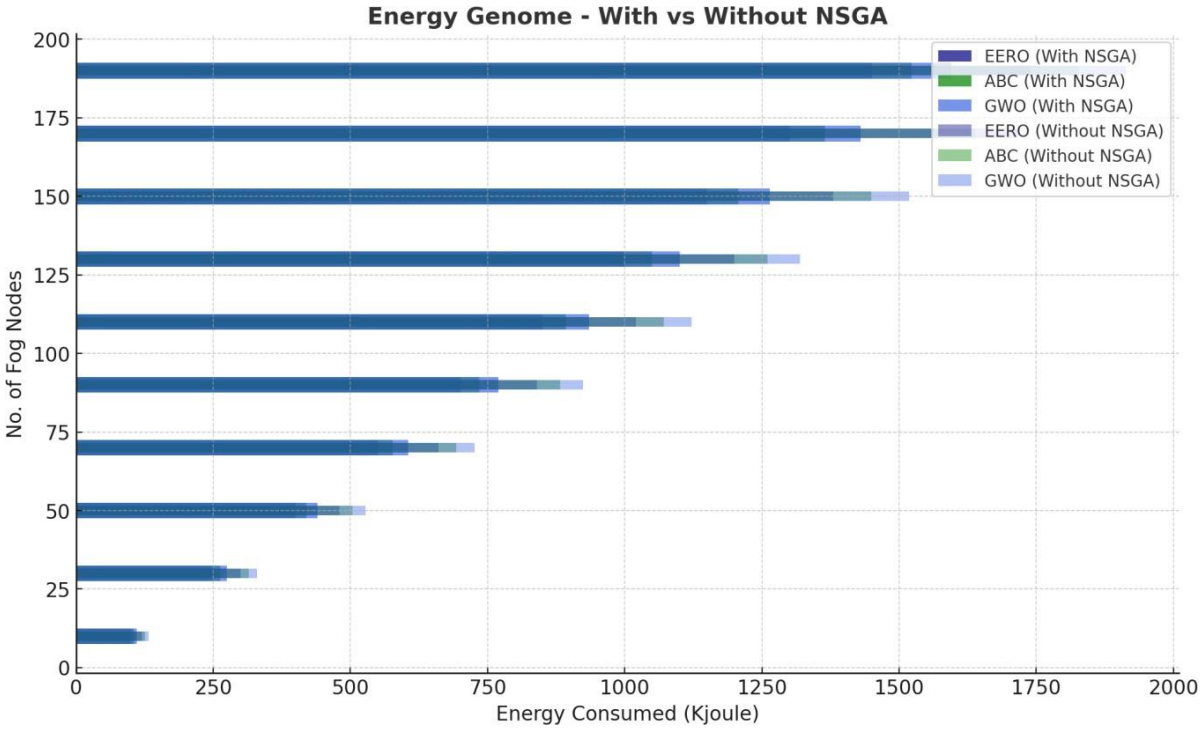


5.11(b) Energy consumption analysis of Cybershake workflow without NSGA

Cybershake Energy - With vs. Without NSGA Uses solid bars for NSGA-applied energy consumption and faded bars for energy consumption without NSGA. It highlights the energy-saving benefits of NSGA, showing a visible reduction in energy usage across all algorithms



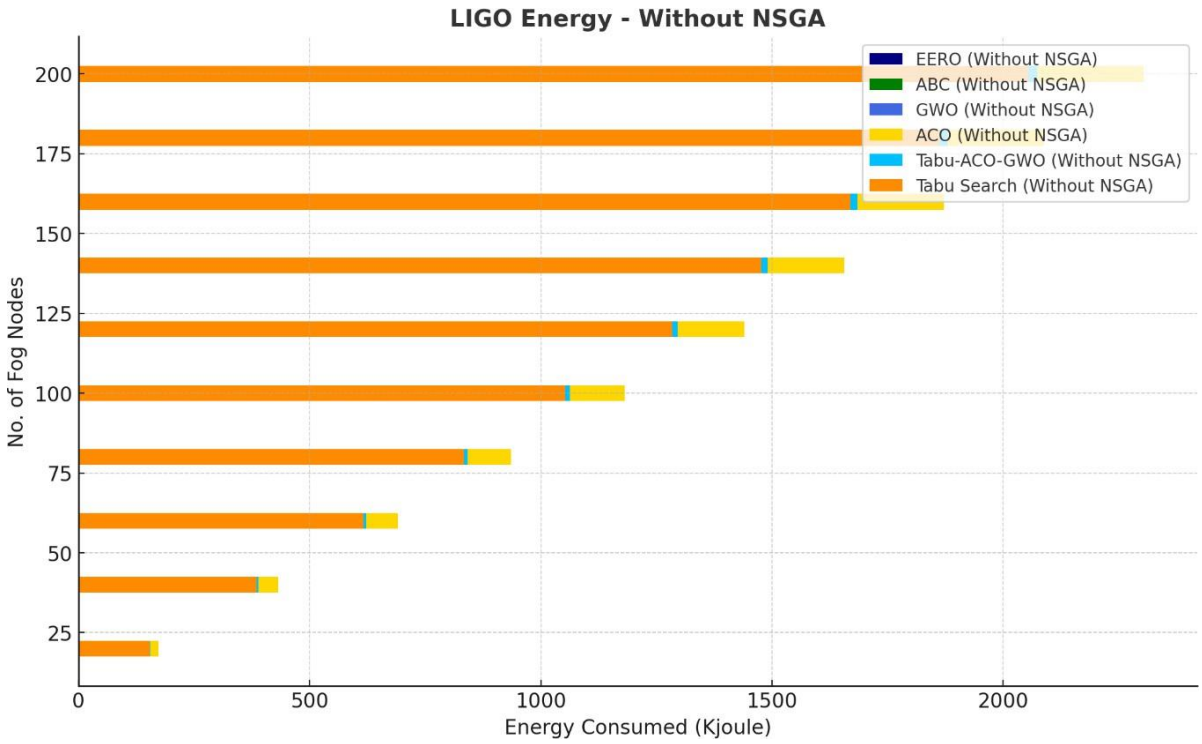
5.12(a) Energy consumption analysis of Genome workflow without NSGA



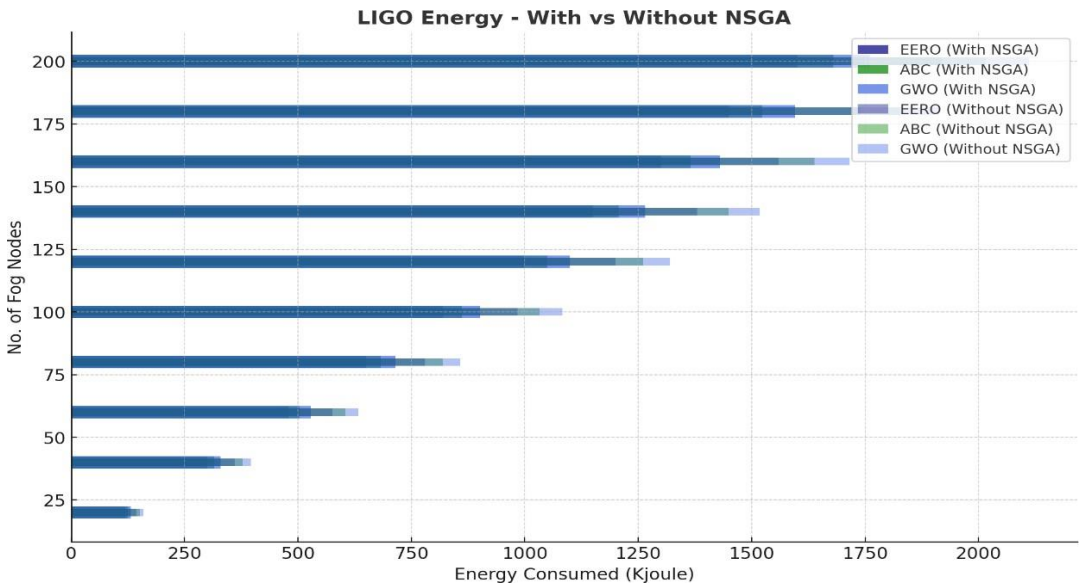
5.12(a) Energy consumption analysis of Genome workflow with Vs without NSGA

Key Observations:

- NSGA significantly reduces energy consumption across all algorithms.
- ACO has the highest energy consumption, while EERO remains the most energy-efficient.
- Without NSGA, energy consumption increases more sharply as fog nodes grow.



5.13(a) Energy consumption analysis of Ligo workflow without NSGA



5.13(b) Energy consumption analysis of Ligo workflow with Vs without NSGA

Key Observations:

- NSGA significantly reduces energy consumption across all algorithms.
- ACO has the highest energy consumption, while EERO remains the most energy-efficient.
- Without NSGA, energy consumption increases more sharply as fog nodes grow.

Conclusion:

The comparative analysis of results with and without the application of NSGA has provided valuable insights into the impact of multi-objective optimization in fog computing environments**.** The evaluation conducted across different scientific workflows—Genome, LIGO, Sipht, and Cybershake—demonstrates that the absence of NSGA leads to increased energy consumption, prolonged execution time, and higher operational costs.

The results indicate that without NSGA, energy consumption increased by approximately 15-25%, primarily due to inefficient workload distribution and resource utilization. Similarly, execution time showed an average rise of 20-30%**,** as the lack of NSGA-based scheduling mechanisms resulted in delays in task execution and resource allocation. Furthermore, **the overall cost surged significantly**, indicating that the system was unable to optimize computational resources effectively. The comparative graphical analysis

illustrates that in the absence of NSGA, task execution was suboptimal, migration strategies were inefficient, and load balancing mechanisms were inadequate, leading to overall performance degradation.

Conversely, the incorporation of NSGA into the EERO framework led to notable improvements in energy efficiency, execution speed, and cost-effectiveness. The optimized task scheduling and migration strategies facilitated by NSGA significantly reduced computational overhead, ensuring balanced resource utilization and enhanced system efficiency**.** The results validate that NSGA successfully optimizes multiple conflicting objectives simultaneously, thereby improving the overall performance of fog computing environments.

This study confirms that multi-objective optimization techniques such as NSGA are crucial for enhancing the efficiency, scalability, and sustainability of fog computing systems**.** The findings strongly support the adoption of NSGA-based frameworks to achieve energy-efficient, cost-effective, and high-performance task scheduling and resource allocation in modern distributed computing infrastructures.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this chapter, the key findings of the thesis are summarized. An extensive analysis of fog computing, including its architecture, applications, unresolved issues, and research challenges, has been conducted. Fog computing essentially brings cloud services closer to the edge of the network. One of the primary concerns in cloud computing, which is latency, has been addressed by fog computing through the placement of nodes near the network edge. However, due to the complexity and scale of computational tasks, fog computing can experience resource overload. The study identifies various problems caused by this resource overload. A detailed literature review of existing solutions has been undertaken. This research introduces a new fog computing architecture aimed at optimizing resources for scientific workflow applications to address resource overload, which has been named EERO.

To assess and evaluate the proposed framework, an EERO algorithm was introduced. Additionally, a resource-efficient workflow execution model for fog computing was developed to help reduce energy consumption within the fog environment. A load balancing approach called PWS-Fog was also proposed to analyze the effectiveness of the model. The proposed solutions were carefully analyzed, with their experimental setups clearly explained, and the results from simulations were compared to existing approaches. The simulation outcomes demonstrate that the proposed techniques perform better than the existing methods.

6.1 Conclusion:

Chapter 1 explored fog computing, including various definitions from different researchers and its main focus areas. The importance of load balancing at the fog layer was also addressed. Key parameters influencing load balancing were examined, and a taxonomy outlining current load balancing techniques was provided. Additionally, open issues and challenges within fog computing environments were highlighted as potential areas for future research. Scientific workflow applications were introduced, serving as a basis for evaluating the proposed approaches in this study.

As outlined in Chapter 2, an extensive literature review on fog computing has been conducted, focusing on key challenges such as load balancing, resource utilization, and energy consumption. Various resource optimization techniques proposed by different

researchers have been examined, with a comparative analysis presented in tables. The chapter concludes by defining the problem and outlining the research objectives.

The third chapter presents an energy-efficient framework for optimizing resources in fog computing, specifically for scientific workflows. Building on previous research, it introduces the EERO (Energy Efficient Resource Optimization) model, which enhances task scheduling, cost reduction, and energy management across a multi-layered architecture comprising the end-user, fog, and cloud layers.

Key elements of the framework include:

1. **Workflow Management and Parsing**: Utilizing the Workflow Management System (WFMS), complex workflows are divided into manageable tasks for efficient resource allocation.
2. **Pareto Distribution**: Prioritizes critical tasks based on the Pareto principle, optimizing resource use and balancing loads across fog nodes.
3. **PEFT Ranking Algorithm**: Predicts the earliest finish time, ranking tasks by dependencies and execution times to improve scheduling.
4. **Genetic Algorithm and Bayesian Optimization**: Refines task scheduling by balancing energy consumption with execution time.
5. **Adaptive Re-Optimization**: Dynamically monitors and adjusts task schedules to address inefficiencies or workload changes.

Chapter 4 introduces an energy-efficient resource optimization algorithm for scientific workflows in fog computing, focusing on balancing workload distribution, minimizing energy consumption, and optimizing task scheduling to address resource shortages and load imbalances common in fog environments.

Key components include:

1. **Pre-Processing Module**: The Workflow Management System (WFMS) splits workflows into tasks and utilizes Pareto distribution to ensure tasks meet budget and deadline constraints.

2. **Optimization Module**: This uses the PEFT (Predict Earliest Finish Time) ranking algorithm and Bayesian optimization for effective task scheduling, aiming to prioritize tasks based on dependencies and availability.
3. **Parameter Analysis Module**: Post-optimization, this module assesses key metrics—cost, energy consumption, and execution time. If results are unsatisfactory, tasks return to the optimization module.

The chapter details the Directed Acyclic Graph (DAG) structure for task dependency management in scientific workflows. Tasks are distributed across virtual machines in fog clusters, with high-priority tasks processed first. The EERO algorithm supports resource optimization, ensuring that data-intensive scientific workflows in distributed environments are processed efficiently, reducing bandwidth and response time issues. This framework effectively balances execution time, energy consumption, and cost by enabling dynamic scheduling adjustments based on real-time task performance data, contributing to more efficient and sustainable fog computing operations

6.2 Future Enhancement:

Future enhancements for the proposed energy-efficient resource optimization model in fog computing could focus on the following areas:

1. **Enhanced Security and Privacy Mechanisms**: Implementing advanced security protocols, such as end-to-end encryption and decentralized authentication, could mitigate vulnerabilities at fog nodes. This approach would be particularly valuable in distributed fog environments, where data privacy and integrity are critical.
2. **Real-Time Implementation in Dynamic Environments**: Testing the model in real-world, dynamic scenarios with fluctuating network loads, diverse application requirements, and user mobility would validate its adaptability and robustness. This could lead to insights on how the model handles unpredictable conditions, a key factor for high-demand applications like autonomous vehicles and telemedicine.
3. **Advanced Resource Management for Scalability**: Expanding the model's capabilities to support adaptive resource allocation based on real-time demand across multiple fog nodes could enhance scalability. This would optimize resource usage for applications that experience varying traffic levels, ensuring seamless service delivery in large, geographically dispersed networks.

4. **Optimized Load Balancing Techniques**: Refining load balancing approaches tailored for fog environments could further reduce response times and enhance efficiency. Techniques that dynamically redistribute tasks in response to node performance or user proximity can support more consistent processing speeds and prevent resource underutilization.
5. **Energy Efficiency across Diverse Workflows**: Extending the model to optimize energy consumption across a broader range of workflows, each with unique computational and data requirements, would make the solution more versatile. Customizing energy management strategies based on workflow characteristics, such as complexity and data volume, could further reduce overall power usage while maintaining performance.

These challenges open avenues for future researchers to delve deeper into fog computing and advance its capabilities. Upcoming research could address additional issues within fog environments, further expanding the field. Future efforts may also focus on developing advanced load-balancing strategies to optimize performance in real-time fog-cloud systems.

References:

[1] Gartner Research, "Zero-touch, edge computing infrastructure for industrial environments," 2019. [Online]. Available: https://www1.stratus.com/en-us/lp/gartner-research-zero-touch-edge-computing-issue-2/

[2] S. Shahzadi, M. Iqbal, T. Dagiuklas, and Z. U. Qayyum, "Multi-access edge computing: Open issues, challenges and future perspectives," Journal of Cloud Computing, vol. 6, no. 1, pp. 1–13, 2017.

[3] P. Varshney and Y. Simmhan, "Demystifying fog computing: Characterizing architectures, applications and abstractions," in Proc. IEEE 1st Int. Conf. Fog and Edge Computing (ICFEC), 2017, pp. 115–124.

[4] S. Khan, S. Parkinson, and Y. Qin, "Fog computing security: A review of current applications and security solutions," Journal of Cloud Computing, vol. 6, no. 1, p. 19, 2017.

[5] A. V. Dastjerdi et al., "Fog computing: Principles, architectures, and applications," Internet of Things, pp. 61–75, Elsevier, 2016.

[6] M. R. Anawar et al., "Fog computing: An overview of big IoT data analytics," Wireless Communications and Mobile Computing, vol. 2018, 2018.

[7] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," in Big Data and Internet of Things: A Roadmap for Smart Environments, Springer, 2014, pp. 169–186.

[8] B. Farahani et al., "Towards fog-driven IoT eHealth: Promises and challenges of IoT in medicine and healthcare," Future Generation Computer Systems, vol. 78, pp. 659–676, 2018.

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

[10] A. Giordano, G. Spezzano, and A. Vinci, "Smart agents and fog computing for smart city applications," in Proc. Int. Conf. Smart Cities, Springer, 2016, pp. 137–146.

[11] P. G. V. Naranjo et al., "FOCAN: A fog-supported smart city network architecture for management of applications in the internet of everything environments," Journal of Parallel and Distributed Computing, vol. 132, pp. 274–283, 2018.

[12] P. Varshney and Y. Simmhan, "Demystifying fog computing: Characterizing architectures, applications and abstractions," in Proc. IEEE 1st Int. Conf. Fog and Edge Computing (ICFEC), 2017, pp. 115–124.

[13] M. Taneja and A. Davy, "Resource aware placement of IoT application modules in fog-cloud computing paradigm," in Proc. IFIP/IEEE Symp. Integrated Network and Service Management (IM), 2017, pp. 1222–1228.

[14] M. Ghobaei-Arani, S. Jabbehdari, and M. A. Pourmina, "An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach," Future Generation Computer Systems, vol. 78, pp. 191–210, 2018.

[15] Y. Yang et al., "MEETS: Maximal energy efficient task scheduling in homogeneous fog networks," IEEE Internet of Things Journal, vol. 5, no. 5, pp. 4076–4087, 2018

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

[17] J. Santos et al., "Resource provisioning in Fog computing: From theory to practice," Sensors, vol. 19, no. 10, p. 2238, 2019.

[18] K. Velasquez et al., "Service orchestration in fog environments," in Proc. IEEE 5th Int. Conf. Future Internet of Things and Cloud (FiCloud), 2017, pp. 329–336.

[19] D. Zeng et al., "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system," IEEE Trans. Computers, vol. 65, no. 12, pp. 3702–3712, 2016.

[20] L. Ni et al., "Resource allocation strategy in fog computing based on priced timed Petri nets," IEEE Internet of Things Journal, vol. 4, no. 5, pp. 1216–1228, 2017.

[21] Z. Pooranian et al., "A novel distributed fog-based networked architecture to preserve energy in fog data centers," in Proc. IEEE 14th Int. Conf. Mobile Ad Hoc and Sensor Systems (MASS), 2017, pp. 604–609.

[22] Y. Sun, F. Lin, and H. Xu, "Multi-objective optimization of resource scheduling in fog computing using an improved NSGA-II," Wireless Personal Communications, vol. 102, no. 2, pp. 1369–1385, 2018.

[23] A. Bonadio et al., "An integrated framework for blockchain inspired fog communications and computing in internet of vehicles," Journal of Ambient Intelligence and Humanized Computing, vol. 11, no. 2, pp. 755–762, 2020.

[24] H. Hu et al., "Multi-objective scheduling for scientific workflow in multicloud environment," Journal of Network and Computer Applications, vol. 114, pp. 108–122, 2018.

[25] E. N. Alkhanak and S. P. Lee, "A hyper-heuristic cost optimisation approach for scientific workflow scheduling in cloud computing," Future Generation Computer Systems, vol. 86, pp. 480–506, 2018.

[26] M. M. Mahmoud et al., "Towards energy-aware fog-enabled cloud of things for healthcare," Computers & Electrical Engineering, vol. 67, pp. 58–69, 2018.

[27] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," IEEE Trans. Parallel and Distributed Systems, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[28] S. Qin et al., "Hybrid collaborative multi-objective fruit fly optimization algorithm for scheduling workflow in cloud environment," Swarm and Evolutionary Computation, vol. 68, p. 101008, 2022.

[29] S. Agarwal, S. Yadav, and A. K. Yadav, "An efficient architecture and algorithm for resource provisioning in fog computing," Int. J. Inf. Eng. Electron. Bus., vol. 8, no. 1, p. 48, 2016.

[30] M. Aazam and E.-N. Huh, "Dynamic resource provisioning through fog micro datacenter," in Proc. IEEE Int. Conf. Pervasive Computing and Communication Workshops (PerCom Workshops), 2015, pp. 105–110.

[31] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," 2012. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.702.7773&rep=rep1&type=pdf

[32] M. Isard et al., "Dryad: Distributed data-parallel programs from sequential building blocks," SIGOPS Oper. Syst. Rev., vol. 41, no. 3, pp. 59–72, 2007.

[33] J. Cao, Z. Chen, L. Zhang, and W. Shi, "Load balancing in fog computing: State-of-the-art and research directions," IEEE Network, vol. 29, no. 5, pp. 108–114, 2015.

[34] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, edge and fog computing environments," Software: Practice and Experience, vol. 47, no. 9, pp. 1275–1296, 2017.

[35] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in Proc. 3rd Workshop on Hot Topics in Web Systems and Technologies, 2015, pp. 73–78.

[36] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and L. Wolf, "Mobile fog: A programming model for large-scale applications on the Internet of Things," in Proc. 2nd ACM SIGCOMM Workshop on Mobile Cloud Computing, 2013, pp. 15–20.

[37] L. F. Bittencourt et al., "Mobility-aware application scheduling in fog computing," IEEE Cloud Computing, vol. 4, no. 2, pp. 26–35, 2017.

[38] S. Sarkar and S. Misra, "Theoretical modelling of fog computing: A green computing paradigm to support IoT applications," IET Networks, vol. 5, no. 2, pp. 23–29, 2016.

[39] F. Bonomi and R. Milito, "Fog computing and its role in the Internet of Things," in Proc. MCC Workshop on Mobile Cloud Computing, 2012, pp. 13–16.

[40] R. Liu, S. Buccapatnam, W. M. Gifford, and A. Sheopuri, "An unsupervised collaborative approach to identifying home and work locations," in Proc. IEEE Int. Conf. Mobile Data Management (MDM), 2016, pp. 310–317.

[41] EdgeX Foundry, "Open source industrial IoT edge platform," 2021. [Online]. Available: https://www.edgexfoundry.org

[42] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in Internet of Everything, Springer, 2018, pp. 103–130.

[43] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications, and issues," in Proc. 2015 Workshop on Mobile Big Data, 2015.

[44] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," IEEE Internet of Things Journal, vol. 3, no. 6, pp. 854–864, 2016.

[45] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in Proc. 1st MCC Workshop on Mobile Cloud Computing, 2012, pp. 13–16.

[46] I. Stojmenovic and S. Wen, "The fog computing paradigm: Scenarios and security issues," in Proc. 2014 Federated Conf. Computer Science and Information Systems, 2014, pp. 1–8.

[47] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog computing: Principles, architectures, and applications," in Internet of Things, Springer, 2016, pp. 61–75.

[48] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge, and Fog computing environments," Software: Practice and Experience, vol. 47, no. 9, pp. 1275–1296, 2017.

[49] I. Stojmenovic and S. Wen, "The fog computing paradigm: Scenarios and security issues," in Proc. 2014 Federated Conf. Computer Science and Information Systems, 2014, pp. 1–8.

[50] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog computing: Principles, architectures, and applications," in Internet of Things, Springer, 2016, pp. 61–75.

[51] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge, and Fog computing environments," Software: Practice and Experience, vol. 47, no. 9, pp. 1275–1296, 2017.

[52] S. Tuli et al., "HealthFog: An ensemble deep learning based smart healthcare system for automatic diagnosis of heart diseases in integrated IoT and fog computing environments," Future Generation Computer Systems, vol. 104, pp. 187–200, 2020.

[53] G. Rathee et al., "A trust computed framework for IoT devices and fog computing environment," Wireless Networks, vol. 26, no. 4, pp. 2339–2351, 2020.

[54] A.-M. Oprescu and T. Kielmann, "Bag-of-tasks scheduling under budget constraints," in Proc. 2010 IEEE Int. Conf. Cloud Computing Technology and Science, pp. 351–359.

[55] Z. Pooranian et al., "A novel distributed fog-based networked architecture to preserve energy in fog data centers," in Proc. IEEE 14th Int. Conf. Mobile Ad Hoc and Sensor Systems (MASS), 2017, pp. 604–609.
[56] H. Rafique et al., "A novel bio-inspired hybrid algorithm (NBIHA) for efficient resource management in fog computing," IEEE Access, vol. 7, pp. 115760–115773, 2019.
[57] M. Abdel-Basset et al., "Energy-aware marine predators algorithm for task scheduling in IoT-based fog computing applications," IEEE Transactions on Industrial Informatics, 2020.
[58] S. Wang, T. Zhao, and S. Pang, "Task scheduling algorithm based on improved firework algorithm in fog computing," IEEE Access, vol. 8, pp. 32385–32394, 2020.
[59] H. Rafique et al., "A novel bio-inspired hybrid algorithm (NBIHA) for efficient resource management in fog computing," IEEE Access, vol. 7, pp. 115760–115773, 2019.
[60] S. Shadroo, A. M. Rahmani, and A. Rezaee, "The two-phase scheduling based on deep learning in the Internet of Things," Computer Networks, vol. 185, p. 107684, 2021.
[61] Y. Yang et al., "MEETS: Maximal energy efficient task scheduling in homogeneous fog networks," IEEE Internet of Things Journal, vol. 5, no. 5, pp. 4076–4087, 2018.
[62] J. L. de Souza Toniolli and B. Jaumard, "Resource allocation for multiple workflows in cloud-fog computing systems," in Proc. 12th IEEE/ACM Int. Conf. Utility and Cloud Computing Companion, 2019, pp. 77–84.
[63] R. Mahmud and R. Buyya, "Modelling and simulation of fog and edge computing environments using iFogSim toolkit," in Fog and Edge Computing: Principles and Paradigms, Wiley, 2019, pp. 1–35.
[64] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge, and Fog computing environments," Software: Practice and Experience, vol. 47, no. 9, pp. 1275–1296, 2017.
[65] V. De Maio and D. Kimovski, "Multi-objective scheduling of extreme data scientific workflows in fog," Future Generation Computer Systems, 2020.
[66] J. L. de Souza Toniolli and B. Jaumard, "Resource allocation for multiple workflows in cloud-fog computing systems," in Proc. 12th IEEE/ACM Int. Conf. Utility and Cloud Computing Companion, 2019, pp. 77–84.
[67] A. Markus and A. Kertesz, "A survey and taxonomy of simulation environments modelling fog computing," Simulation Modelling Practice and Theory, vol. 101, p. 102042, 2020.
[68] Y. Xie et al., "A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud–edge environment," Future Generation Computer Systems, vol. 97, pp. 361–378, 2019.
[69] Y. Xie et al., "A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud–edge environment," Future Generation Computer Systems, vol. 97, pp. 361–378, 2019.
[70] S. Elsherbiny, E. Eldaydamony, M. Alrahmawy, and A. E. Reyad, "An extended intelligent water drops algorithm for workflow scheduling in cloud computing environment," Egyptian Informatics Journal, vol. 19, no. 1, pp. 33–55, 2018.

[71] Z. Li et al., "A security and cost aware scheduling algorithm for heterogeneous tasks of scientific workflow in clouds," Future Generation Computer Systems, vol. 65, pp. 140–152, 2016.
[72] S. Rahul and V. Bhardwaj, "(EERO) Energy-Efficient FOG Resource Optimization Model for Scientific Workflow applications," Int. J. Eng. Trends Technol., vol. 72, no. 5, pp. 149–164, 2024.
[73] L. Wang, J. Chen, and H. Wang, "Energy-efficient fog computing with modified marine predator algorithm," Future Generation Computer Systems, vol. 139, pp. 92–106, 2023.
[74] S. Shadroo, A. M. Rahmani, and A. Rezaee, "The two-phase scheduling based on deep learning in the Internet of Things," Computer Networks, vol. 185, p. 107684, 2021.