Designing Cost-Latency Optimization of Task Scheduling Algorithm Over Fog-Cloud Computing Environments

Gadissa Dagne Binagde

A Thesis Submitted to the Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

September 2022

Adama, Ethiopia

Designing Cost-Latency Optimization of Task Scheduling Algorithm Over Fog-Cloud Computing Environments

Gadissa Dagne Binagde

Advisor Dr. Ing Frezewd Lemma

A Thesis Submitted to the Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in Partial Fulfillment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

September 2022

Adama, Ethiopia

Declaration

I declare that this thesis entitled "**Designing cost-latency optimization of task scheduling algorithm over fog-cloud computing environments"** is my own work and has not been submitted to any university for a similar purpose. The references used in this thesis are duly recognized by proper citations.

Gadissa Dagne Binagde	_____	_____
Name of student	Signature	Date

Approval Page of M.Sc. Thesis

I, the advisors of the thesis entitled "**Designing cost-latency optimization of task scheduling algorithm over fog-cloud computing environments**" and developed by Gadissa Dagne Binagde hereby certify that the recommendation and suggestions made by the board of examiners are appropriately incorporated into the final version of the thesis.

Dr. Ing Frezewd Lemma _____ _____

Major Advisor Signature Date

Approval of Board of Reviewers

We, the undersigned, members of the Board of Examiners of the thesis by Gadissa Dagne Binagde have read and evaluated the thesis entitled" **Designing cost-latency optimization of task scheduling algorithm over fog-cloud computing environments**" and examined the candidate during open defense. This is, therefore, to certify that the thesis is accepted for partial fulfillment of the requirement of the degree of Master of Science in Computer Science and Engineering.

_____	_____	_____
Chairperson	Signature	Date
_____	_____	_____
Internal Examiner	Signature	Date
_____	_____	_____
External Examiner	Signature	Date

Finally, approval and acceptance of the thesis is contingent upon submission of its final copy to the Office of Postgraduate Studies (OPGS) through the Department Graduate Council (DGC) and School Graduate Committee (SGC).

_____	_____	_____
Department Head	Signature	Date
_____	_____	_____
School Dean	Signature	Date
_____	_____	_____
Office of Postgraduate Studies, Dean	Signature	Date

Table of Contents

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATION

LIST OF ABBREVIATION AND ACRONYMS

BLA	Bee Life Algorithms
CISCO	Commercial and Industrial Security Corporation
FCFS	First Come First Served
GA	Genetic Algorithm
GWO	Grey Wolf Optimizer
IACO	Improved Ant Colony Optimization
ICT	Information and Communication Technologies
IoT	Internet of Things
IHGWO	Improved Hybrid Grey Wolf Optimizer
IPSO	Improved Particle Swarm Optimization
MGWO	Modified Grey Wolf Optimization
NIST	National Institute of Standards and Technology
PSO	Particle Swarm Optimization
QoS	Quality of Services
RR	Round Robin
SA	Simulated Annealing
SJF	Short Job First
SLA	Service Level Agreement
TCaS	Time-Cost Aware Scheduling

ACKNOWLEDGEMENT

First and foremost, I want to express my gratitude to God for giving me the time and strength to finish this thesis. next to that, I want to thank my advisor, Dr. Ing. Frezwed Lemma, from the bottom of my heart. He helped me succeed in this research with his encouragement, direction, passion, and willingness to share his enormous expertise. I also want to thank my parents for always encouraging and supporting me as I worked on this thesis. Next, I want to thank all of my closest friends for helping me with this job by contributing their knowledge and suggestions. Finally, I want to thank my beloved brother Higu Dagne for his consistent encouragement and support in helping me finish my research.

ABSTRACT

The Internet of Things has recently generated large amounts of data, which has an impact on cloud computing infrastructures. Fog computing is an extension and complement to cloud computing in which services like computation and storage are provided at the network's edge. However, the variation in fog resource capacity has introduced new challenges, such as latency reduction, deadline meeting, and cost reduction, in order to efficiently utilize resources and provide services. Task scheduling is essential for assigning tasks uploaded from IoT devices to efficient fog and cloud nodes while accounting for latency, resource utilization, deadlines, and costs. As a result, if proper task allocation to available resources is not strictly followed, it results in wasted resources and unsatisfactory service quality. Over the years, different approaches such as task scheduling and resource allocation have been proposed to optimize the quality of service in a fog-cloud environment. However, it is still challenging to get an accurate estimate of the optimum quality of services such as latency, resource utilization, deadline, and cost. This study proposed an improved hybrid grey wolf optimization algorithm for allocating IoT device-generated tasks to appropriate cloud and fog nodes. The suggested approach modifies the goal function to handle multiple objectives into a single objective using the scalarization method in order to solve the task scheduling problem that includes latency and cost while taking the deadline of tasks into account. On the cloudsim simulator, the effectiveness of the proposed approach has been evaluated. Based on experimental results, the proposed algorithm has reduced latency by 22.45%, cost reduction by 17.67%, outperforming FCFS, and has reduced latency by 21.9%, cost reduction by 18%, outperforming SJF algorithms. It has also reduced latency by 4 %, cost reduction by 7.1% outperforming PSO, reduced latency by 4.4%, cost reduction by 6.7%GWO, and reduced latency by 5%, cost reduction by 6.65 % outperforming GA. Additionally, compared to existing algorithms, the suggested approach more successfully meets the requirement of a task deadline.

***Keywords**: Cost-Latency, Task Scheduling, Fog Computing, Cloud Computing, Internet of Things*

CHAPTER ONE

1. Introduction

1.1. Background of the Study

One of the biggest changes in information and communication technology (ICT) today is the Internet of Things, which now powers many of the daily activities of people, including surveillance, businesses, transportation, and smart cities. Due to the need for computational resources for real-time processing, several delay-sensitive tasks have therefore emerged(IDC, 2016).

To address this need IoT requires computation and storage services to complete its specified task from high-level computing resources such as fog computing and cloud computing. Cloud computing delivers computing resources such as hardware, operating systems, storage capacity, and software applications as needed. It's centralized provisioning of services for all end-device requests, with all services provided by a central data center farm. Due to the distance between cloud servers and user devices, cloud computing faces issues such as excessive latency, large server loads, lack of mobility support, high communication costs, and redundancy of data on servers. To resolve these limitations, Cisco introduced a new computing environment called fog computing (Liu et al., 2017b).

Fog computing is a type of cloud computing that provides services such as storage, computing, and applications locally between the cloud and the data source(Mahmud et al., 2018).It makes use of computing-enhanced network components, such as smart routers, proxy servers, switches, and base stations, that can do computations, store data, and link to other nodes. Edge network equipment such as routers, switches, and base stations are more capable of processing and storing data from IoT devices as software and hardware technologies advance(Sreenu & Sreelatha, 2019).

According to a 2019 study by the International Data Corporation (IDC), by 2025, there will be 41.7 billion IoT devices in the world, generating 79.4 zettabytes (ZB) of data. These IoT devices may require demanding quality of service requirements, such as latency sensitivity, which can be challenging to self-manage due to resource constraints(IDC, 2016).

For this reason, fog devices have emerged as a near-future computing resource that can handle low-latency data transfers for various IoT-enabled systems by achieving quality of service (QoS). Fog computing moves the required computing resources and services from the central

cloud to the edge of the network to reduce the latency requirements of IoT devices. However, the performance of fog nodes can be poor in computationally intensive tasks due to their resource constraints for huge data volumes(Hao et al., 2017).

Many IoT-based computing applications have been developed for systems that can face deadlines in areas such as healthcare, transportation, augmented reality, and gaming. To overcome this challenge, appropriate task scheduling is needed in fog and cloud nodes to ensure the efficient quality of service, such as meeting real-time application deadlines, reducing latency, and reducing the cost associated with task assignment.

Therefore, efficient and effective task scheduling policies need to be addressed in order to derive significant benefits from fog devices. On the other hand, task scheduling is an NP-hard problem that requires an optimal solution to implement task assignments efficiently. Thus, task scheduling in fog cloud nodes requires efficient planning to minimize delay and meet deadlines, minimize costs and maximize resource utilization. The cost and latency problems have become a critical requirement in fog nodes as the number of IoT devices increases. In this work, we have proposed designing cost-latency optimization task scheduling over fog-cloud computing environments based on a meta-heuristic nature inspired algorithm that considers latency minimization, cost minimization, and deadlines.

The meta-heuristic search algorithm is well-known for finding optimal solutions to NP-hard problems such as scheduling(Chakraborty & Kar, 2017). Many metaheuristic approaches proposed in recent years have addressed task scheduling problems with multiple objectives as a single goal(Zhong et al., 2016)(Natesan & Chokkalingam, 2020). These approaches seek a single optimal solution. A few researchers have focused on using a metaheuristic algorithm to find near-optimal solutions and provide trade-offs to cloud service providers by taking multiple objectives into account at the same time(Alsadie & Member, 2021).

In this study, we proposed designing cost-latency optimization task scheduling using a meta-heuristic improved hybrid grey wolf optimization algorithm to find a best or nearly best solution while simultaneously taking into account multiple conflicting objectives such as latency and cost with taking into account task deadline.

1.2. **Motivation**

Regardless of the fact that it is a modern concept in information and communication technology, fog and cloud computing are becoming increasingly popular in the research world. The

allocation of resources in cloud computing has a significant impact on work performance, availability, and responsiveness. Because cloud workloads are dynamic, providers must use a good scheduling algorithm to make their system run quickly with few resources and a short response time.

Latency-sensitive applications, such as healthcare and gaming, require real-time data processing to deliver the expected performance(Naha et al., 2018). Latency-insensitive big data processing, on the other hand, necessitates lower latency and overhead to meet the needs of resource providers. Furthermore, the heterogeneous nature of Fog and Cloud resources is critical to evenly distributing workloads from IoT devices. Fog computing can solve latency problems, but because fog resources vary in processing and storage capabilities, efficient tasks must be assigned to each node. As a result, resource utilization, latency minimization, deadline and cost minimization must be ensured to meet the needs of both users and resource providers, which is still difficult to achieve.

1.3. Statement of the Problem

Big data processing and an increase in IoT applications have led to the emergence of more complex time-sensitive applications than in traditional computing. Today, processing and responding to each request in a timely manner is quite difficult due to the billions of IoT devices that have developed and the enormous amount of data they have produced and requesting fog and cloud computing. With the growth of the Internet of Things (IoT), fog computing is faced with difficult challenges related to resource restrictions, such as improving quality of service (QoS), lowering latency, lowering costs, and the user's need to meet task deadlines in order to finish a request.

The most common cause of the degradation of this service quality in fog computing is the lack of appropriate task scheduling, which leads to low resource utilization and an unsatisfactory quality of service, creating high latency for latency-sensitive requests. If tasks are not properly scheduled, resource constraints in fog computing, such as processing and storage limitations, can increase latency for most real-time applications and increase costs. Therefore, to meet the requirement of these applications, designing a task scheduling algorithm that defines where to assign tasks and when to assign them on the available node is inevitably required. Many researchers have tried to design task scheduling algorithms to achieve the quality of service

in a fog-cloud environment without considering deadlines, latency, and costs at the same time(Nguyen et al., 2019)(Hoang & Dang, 2017).

Proper task assignment on nodes is required to efficiently utilize resources. Most of the existing task scheduling methods (J. Wang & Li, 2019)(Kabirzadeh et al., 2018)(Alsadie & Member, 2021)for fog computing assign tasks on an available virtual machine without considering task size and deadline of tasks. The decisive decision of which task needs to be executed on which node is a challenging problem that needs to be addressed. Hence, task scheduling algorithms need to be designed that can determine latency-sensitive, latency-less sensitive, and latency-insensitive as well as allocate tasks into small, medium, and large tasks and assigning them to effective nodes while taking latency and cost into account.

In this research, we have designed a cost-latency optimization task scheduling in fog-cloud computing environment using an improved hybrid grey wolf optimization algorithm which classifies tasks based on deadline and task-size threshold values and assigns them to latency and cost-efficient nodes, ensuring the task's deadline. This approach minimizes latency, meets deadline requirements, and reduces costs.

1.4. Research Questions

This thesis intends to answer the following three research questions:

RQ1: How do optimize cost-latency together to have efficient task scheduling in fog-cloud computing environments?

RQ3: What are the different parameters that can be used to classify tasks?

RQ4: What fog architecture should be used to implement the proposed task scheduling algorithm in a fog-cloud environment?

1.5. The Objective of the Research

1.5.1. General Objective

The general objective of the proposed study is to design cost-latency optimization of task scheduling algorithm over the fog-cloud computing environments

1.5.2. Specific Objective

To accomplish the objective of general the proposed consider, the specific goals are

- Designing a strategy for assigning tasks to the low-level fog nodes, upper-level fog nodes, and cloud nodes in order to reduce latency, meet deadlines, and reduce costs.

- ❖ Identifying a suitable fog-cloud architecture that meets the latency and cost optimization task scheduling algorithm requirements.
- ❖ Identifying a suitable simulator for experimentation based on the proposed task scheduling requirements over the fog-cloud environment.
- ❖ Comparing the latency and cost optimizing task scheduling algorithm's performance to existing approaches.

1.6. Significance and beneficiaries of the Study

1.6.1. Significance of the Study

Scheduling is critical because it affects resource consumption and overall system performance. The proposed research has benefited all fog and cloud computing users, particularly fog and cloud service providers. The proposed technique would provide the user with quick and flexible access to and use fog computing resources. It has enabled the cloud service provider to meet the needs of clients in accordance with SLAs. In other words, if a task scheduling method is impractical, the service provider may be unable to meet the expectations of the customer.

This research work is beneficial to improving service quality in areas such as task completion, resource utilization, cost reduction, and latency reduction. This work is being used to avoid the complexity caused by real-time fog. This research study employs resources found near-end devices to minimize latency, lower costs, and meet task deadlines, as well as cooperative high-level fog nodes with cloud nodes to meet tasks with deliberate deadlines.

1.6.2. Beneficiaries of the study

The beneficiaries of the research are the following:

- ✓ **The fog-cloud end-users**: fog and cloud end users can access various cloud and fog resource through internet within a minimum time and pays as their use. A good scheduling algorithm will satisfy user requirements by improving the latency and minimizing the cost.
- ✓ **The fog-cloud providers**: providers can easily manage a fog and cloud resource and they become more cost effective due to optimal scheduling of a task and balancing the entire load of the system.

- ✓ **Future researchers and academicians:** The proposed algorithm can be used by researchers and academicians for further optimization. It is also used at the university level for research and as an academic reference.

1.7. Expected outcome of the Study

Finally, this work should satisfy end-users and service providers by reducing the cost-latency ratio and properly scheduling tasks in the fog and cloud nodes. The result of this work has been to optimize the cost-latency of task scheduling while meeting deadlines.

1.8. Scope and Limitations

The research's primary goal is to create a cost-delay optimized task scheduling algorithm that minimizes delays, ensures task completion on time, and reduces costs. This work was motivated by task categorization, a method for categorizing tasks, and the discovery of suitable nodes in a fog-cloud environment that meet the goals of latency reduction, on-time completion, and cost reduction. The study excludes fault tolerance, and energy consumption as limitations.

1.9. Methodology of the Study

In conducting this research, we thoroughly examined existing knowledge in this area. Book reviews, journal articles, research papers, topic reports, and other online documents were reviewed to gain insight into previously studied methods and approaches to task scheduling in fog and cloud computing environments.

Task scheduling in fog and cloud computing environments should lead to improvements in quality of service (QoS) in terms of latency, cost, timing, and resource utilization. In general, the techniques and methods described below have been used to achieve the above objectives and answer the research question.

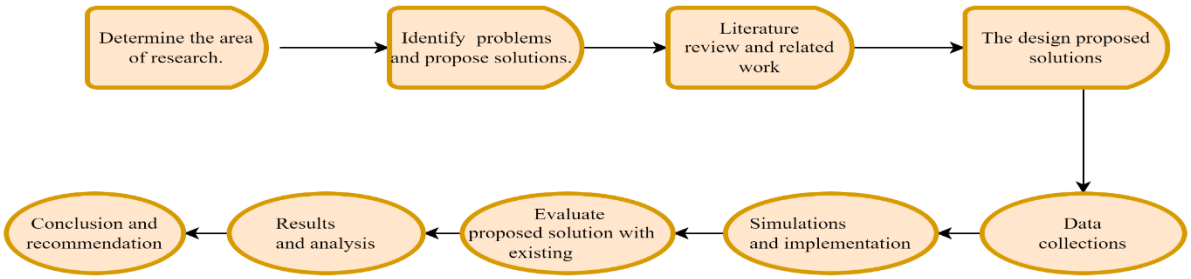

Figure 1.1 Procedure of the Study

1.9.1. Implementation and Design tools

NetBeans, a development environment, has been chosen as the best tools for implementing the proposed solution. NetBeans includes a Java Development Kit for Java developer, and as a result, the cloudsim framework package, which is built by Java, is integrated for simulation, as well as Edraw Max as a design tool. Edraw is a design tool that allows software developers to create diagrams such as flowcharts, workflows, and UML diagrams. In the research, this tool is used to design and draw diagrams such as flowcharts and workflows.

Hardware Requirements

The hardware requirements for conducting this research are listed below.

- ✓ Processor: Intel(R) Core (TM) i3-3120M CPU @ 2.50GHz.
- ✓ RAM: Installed memory (RAM) 8.00GB.
- ✓ System Type: x64 based processor.
- ✓ Hard Disk: 1000 GB

1.10. Thesis Organization

Following is how the remaining portions of this study are structured: In addition to a thorough explanation of the concepts of cloud computing, fog computing, and Internet of Things devices, the second chapter looks at the literature review and related research that have already been done on the fog-cloud environment in relation to task scheduling. Chapter Three provides a detailed explanation of the proposed solution's design as well as the architecture adapted to satisfy the design requirements. It looks into the process for categorizing tasks and putting them in the right nodes to raise the quality of services. Services latency, which are utilized to locate suitable nodes, make up this system. The Chapter Four compares the proposed algorithm to existing approaches in terms of cost, deadline, and latency minimization using the Cloudsim simulator. In Chapter Five, the conclusion, recommendation, and future works of the research work have been discussed.

CHAPTER TWO

2. Literature Review and Related Works

2.1. Literature Review

This chapter focuses on the concepts of fog computing, cloud computing, fog computing scheduling, as well as their properties, guiding principles, and uses in service delivery.

Clients can now access services at any time and from any location due to cloud computing. Fog computing, on the other hand, introduced to be a solution for the cloud computing limitations of latency, mobility, and location awareness issues raised on by IoT devices(Liu et al., 2017a). The principles of task scheduling and existing architecture for collaborating IoT devices, fog computing, and cloud computing nodes have been thoroughly investigated.

2.2. Overview of Cloud Computing

Cloud computing is a system that allows users to pay as they go for computing resources such as applications, storage, and processing power over the internet(Vuyyuru et al., 2012). Its goal is to allow users to profit from shared pools of configurable resources without having to worry about the low-level structure of the system. Instead, they can focus on their business problems. Cloud computing is defined as a model for enabling omnipresent, helpful, on-demand network access to a shared pool of programmable figuring assets that can be instantly provided and delivered with minimal management activity or specialist organization connection. The five fundamental characteristics, three service models, and four deployment models included in these norms act as a guide.

These guidelines serve as a road map for the cloud, encompassing five key characteristics, three service models, and four deployment models. Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) are three types of cloud services. Because of its high-quality services and measurable balanced prices, cloud computing has gotten a lot of attention from both the scientific and manufacturing communities. According to the study (Mohammad, 2018), cloud computing resource utilization has increased dramatically in recent years as it has become more widely adopted by businesses and organizations.

2.2.1. **Characteristics of Cloud Computing**

According to NIST(Moghaddam et al., 2015), computing capabilities must meet five major criteria in order to be regarded as a service. Every computational capability must be on-demand, internet-accessible, multitenant, and elastically offered(Moravcik et al., 2018).

1. **On-Demand Services**

On-demand computing service is a service model that provides every computing service based on the user's need of resources. Every service is made accessible on an “as needed” basis.

2. **Broad Network Access**

People need to access information existing anywhere by any device they possess either thin client or thick client platform such as mobile phones, tablets, laptops, desktops, and workstations (Mohammad, 2018). One characteristic of cloud computing is to make its service available over a network and deliver it to the user through heterogeneous platforms. Customers are supposed to get information from cloud computing hosting companies irrespective of their current location and the device they are using.

3. **Resource Pooling**

Multiple customers could be served by sharing pooled resources which leads to low costs as a result of multi-tenant models. The multi-tenant model is an architecture that provides the ability to serve multiple users by a single computing resource but acts as logically independent resources to each user. Resources can be storage, processing, memory and network bandwidth that the customer has no knowledge of their exact locations but can identify the country or state they are available as high-level abstraction.

4. **Rapid Elasticity**

Computing capability is provided in some cases automatically according to variable user demands. Dynamically allocating resources is to either shrink or grow computing capabilities based on consumer demand which enables it to appear to be unlimited. In older systems lack of resources such as storage and memory are often easily detectable to the consumer when their capacity reaches maximum limit.

5. **Measured Service**

Transparency of resource past usage by the user needs to be apparent for both resource supplier and the consumer based on metered services. Utilized services can be measured so that consumers have to pay for only services they use which in turn permit the supplier to

automatically optimize and scale its resources based on needs. NIST stated that resources that are in use need to be transparent, visible and measurable to both resource provider and user of the resource so that the customer pays for the only resource they use.

2.2.2. Cloud Service Delivery Model

Cloud services are services that will be provisioned to customers over the internet allowing easy access to applications, storage, and processing resources without deep knowledge of internal infrastructure (Karagiannis, 2011). Services given by the cloud are more helpful in reducing cost and scaling resources with more flexibility to provide it only on demands. NIST cloud computing definition defines three possible categories of services as PaaS, SaaS, and IaaS(Moravcik et al., 2018).

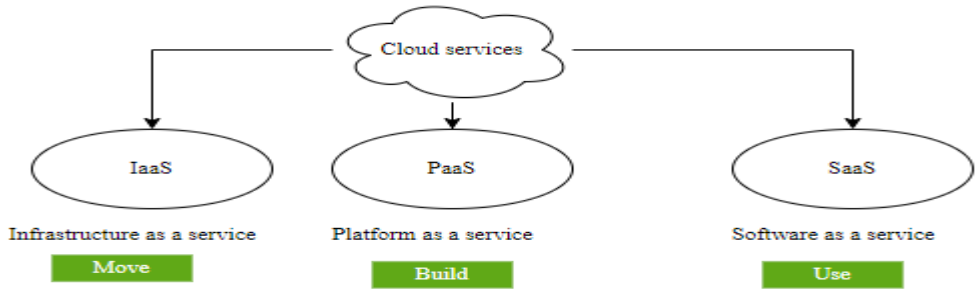

Figure 2. 1 cloud services

1. **Infrastructure as a Service**

Infrastructure as a service is a cloud computing offering that vendor affords a customer access to hardware and software resources that can be storage, networking, servers, and other supporting resources including operating systems which afterward customers run their application and platforms within provided infrastructure(Huang et al., 2019). These allow the client to pay only for infrastructure such as processing and storage that might be scalable depending on demands. Therefore, users need no concern about the cost of buying and maintaining hardware. The user can monitor, manage, deploy and run arbitrary software which can embrace operating systems and other applications but they do not have control over underlying cloud infrastructure.

2. **Software as a Service**

Software as a Service allows the customer to use software applications that reside remotely over cloud infrastructure which means users do not install, configure and upgrade the application on

their local devices but access it through a web browser (e.g., web-based Email) or an API. This removes the cost of hardware purchase, provisioning, and repairs, as well as software configuration, installation, and deployment (Gibson et al., 2012).

3. **Platform as a Service**

In this type of cloud service, cloud providers provide everything that developers need to build an application such as development tools, operating systems, and other infrastructures. All necessary management of tools for development will take place behind the scenes at cloud providers so that developers do not bother to update, install and configure the development environment (Gibson et al., 2012). The services offered by PaaS to its users are operating system, database management system, server software, storage, network access, hosting, tools for design and development (Gibson et al., 2012). Those services are hosted by the cloud and users can access them through the web browser. Moreover, the user can select only the feature they need while rejecting other features that they do not require to include in their applications from preconfigured features given by the provider.

2.2.3. Cloud Deployment Model

Cloud deployment model describes how it operates and who gets access to hosted cloud resources depending on user needs that require a reduction in their capital expenditure. The deployment model represents the category of cloud environment based on user requirement, size, and access as well as describes cloud purpose and nature. There are four types of cloud deployment models such as private cloud for a private organization, public cloud for general users, hybrid cloud, and community cloud(Vuyyuru et al., 2012).

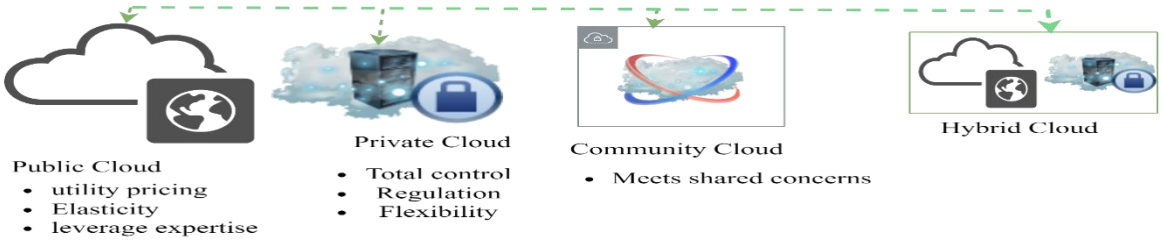

Figure 2. 2 cloud model

1. **Private Cloud**

This type of deployment model allows access to cloud resources only to specific organizations. In a sense, the cloud resource will be guarded by advanced firewalls under the investigation of the IT department that belongs to a specific organization. Cloud resources and infrastructures are exclusively given to the single organization and solely managed by it. Cloud resources can be hosted at organization premises (i.e., onsite private clouds) or at hosting companies (i.e., at cloud providers). Besides that, one benefit of using a private cloud for the organization is that it provides secured and more control over its resources within the boundary of a privileged organization. A private cloud can be deployed at the organization site which in that case higher performance is achieved as a result of the short distance of communication. In contrast, cost and logistical challenges associated with the management of required hardware could lead to the challenging problem (Mell et al., n.d.).

2. **Public Cloud**

The public cloud deployment model avails its cloud infrastructure to the general public and is possessed by an organization that vends services. Resource scale on demand can be realized and offered over an internet connection which will be paid based on a pay-per-usage fee. Also, services are used by the customer through a web browser or other software application (Mohammad, 2018). The resource will be hosted at the cloud provider's site that they manage and pool resources according to the capacity needed by users. Most importantly, the advantages of the public cloud include easy and inexpensive setup, on demand scalability, full technician expertise, and data availability. While in contrast, public clouds are exposed to security and privacy issues since resources are managed by a third party. Some examples of public clouds are: Google Gmail, Amazon AWS, Microsoft Office 365(Mohammad, 2018).

3. **Community Cloud**

A community cloud works where two or more cloud customers have common objectives such as security, privacy, and mission objectives that need to share information. In the same way as other cloud models, it can be controlled by an organization or third party, or a combination of them. Besides this, it could be off premises or on-premises depending on needs.

4. **Hybrid Deployment Model**

A hybrid deployment model is the combination of two or more other deployment models such as the private deployment model, public model, and community model. Most of the time it is a

combination of public cloud and private cloud. Through a hybrid deployment model, services and data from the other model can be integrated for achieving a unified, automated, and secured environment (Karagiannis, 2011).

2.3. Limitation of Cloud Computing

One of the major limitations of cloud computing is the high latency response time to process an application due to distant communication and the centralized nature of cloud computing. Some applications such as robotics, smart transportations, health care, real-time manufacturing, and gaming require a response within milliseconds. Providing such a service for those applications at cloud is impossible for applications to cope with their objectives due to delay (Khanvilkar et al., n.d.).

2.4. Fog Computing

Cisco introduced the term “Fog Computing” as a new model for a decentralized form of cloud computing where services are provided at the edge of the network (Hassan, 2004). It is an approach aimed to help applications that need low latency services(Liu et al., 2017b)fog brings cloud services closer to the things that generate and act on data, such as IoT devices. Fog computing (FC) is a new distributed computing platform that aims to bring computation closer to its data sources, lowering latency and cost of data delivery to a remote cloud. Many Internet-of-Things applications, especially latency-sensitive and latency-insensitive services, might benefit from this capability and its associated benefits(Liu et al., 2017).

Fog computing, According to(Bonomi et al., 2012) is a distributed computer architecture at the network's edge that uses a pool of heterogeneous resources to provide services such as computation and storage to users. Because of its near to end devices, the fog computing layer delivers computing, network, and storage services near the end nodes; similarly, it is a highly distributed computing resource bringing new services that restrict latency concerns to end devices. Some of these services are a scaled version of the ones provided by cloud computing while most of these services emerged recently in response to IoT challenges(Hu et al., 2017). Fog computing is complementary to cloud computing noticeably which means it does not replace cloud computing but develops the computing of cloud services at the very end nodes of the network. Fog nodes primarily provide local processing, as a result, low latency and context awareness challenges of IoT requirements will be solved; on the other hand, Cloud provides

global centralization. However, some applications for instance big data analysis and image processing need both fog localization and cloud globalization to get better quality service(Bonomi et al., 2012).

2.4.1. **Fog Computing Characteristics**

Fog computing is essentially a cloud extension that is closer to the things that function with IoT data. As shown in Figure 2.3, fog computing acts as a bridge between the cloud and end devices, bringing processing, storage, and network resources closer to the end devices. These devices are called fog nodes. They can be set up anyplace there is a network connection. Fog nodes can be industrial controllers, switches, routers, embedded servers, and video surveillance cameras since they include processing, storage, and network connectivity (Sabireen & Neelanarayanan, 2021).

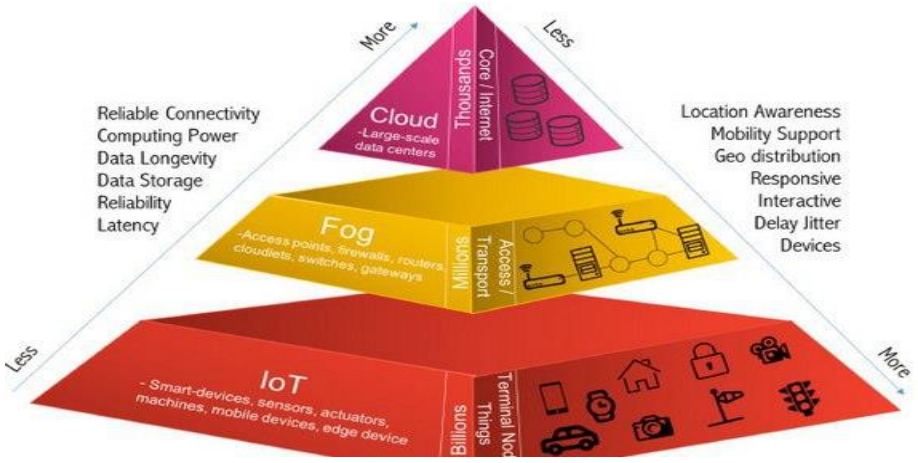

Figure 2. 3 fog computing architecture from(Yousefpour, Fung, Nguyen, Kadiyala, Jalali, Niakanlahiji, Kong, & Jue, 2019)

Cloud Layer

A cloud layer is a packed pool of multiple high-performance servers that can provide services like processing and data storage for comprehensive analysis. Unlike traditional cloud computing, not all task loads must be sent to cloud nodes; rather, resource utilization must be improved by using the service at fog nodes to meet application needs.

Fog Layer

The fog layer is located at the edge of the network and consists of massive fog nodes with computing capabilities such as intelligent routers, gateways, switches, access points, base stations and dedicated fog servers. These nodes can compute and temporarily store delay-sensitive data in a variety of locations, including retail malls, cafes, streets, and parks.

IoT Layer

Sensors, smart vehicles, mobile phones, wearables, readers, and other IoT devices are all part of the IoT layer, which is much closer to the user and physical environment. This layer is responsible for collecting feature data from physical objects and transferring it to the architecture's upper layer for processing or storage. Fog computing is considered to be the building blocks of the cloud. According to Ai et al. (Ai et al., 2017) and (Yi et al., 2015).

The characteristics of fog computing can be summarized as follows.

- **Low latency**: due to its proximity to the end device, fog computing will enable much faster response times and service analysis.
- **Mobility support**: one of the most important aspects of fog applications is their ability to connect directly to mobile devices, enabling mobility technologies such as Locator Identifier Separation Protocol (LISP), which requires a distributed directory system.
- **Real-time interaction**: unlike batch processing used in the cloud, fog computing applications enable real-time interaction between fog nodes.
- **Heterogeneity**: as fog nodes or end devices are developed by different vendors, they have different shapes and sizes and must be distributed according to their platform. Fog can run on different platforms.
- **Low latency and location awareness**: fog computing promotes location awareness by allowing fog nodes to be placed in different locations. In addition, since the fog is closer to the end device, it has lower latency when processing data from the end device.
- **Geographic distribution**: unlike centralized clouds, fog services and applications are distributed and can be deployed anywhere. The distributed computing and storage resources provided by "Fog" are capable of servicing this large volume of endpoints.
- **Interoperability**: Fog components can communicate with other domains and service providers.
- **Support for online analytics and cloud interaction:** Fog sits between the cloud and the end device, helping to absorb and process data close to the end device.

2.4.2. Fog Applications

Smart grid

A smart grid is the power distribution network of the future. Transmission lines, substations, transformers, and other infrastructure are all part of a smart grid. It is an automated and

decentralized energy distribution network that uses bi-directional flows of energy and data. Smart grids provide a clear energy distribution system that allows service providers and customers to monitor and control prices, production, and consumption in real time(Rahman & Wen, 2018). In a big data environment, millions of smart meters are installed in consumers' homes. Fog collectors are used at the edge of the process to collect, process, and filter information at the local level before sending it to a cloud data center for long-term storage(Barik, 2017).

Health care system

Medical services and applications are slow to respond and generate sensitive patient data. The data produced includes sensitive and personal information. Similarly, location data can be sensitive in certain situations. Increased instability and latency can cause a number of problems in medicine and telemedicine applications. In such cases, fog tracking may be an appropriate paradigm in healthcare scenarios (Rahman & Wen, 2018). Fog computing plays an important role in primary care due to the low latency associated with implantable medical devices, ambulance communication and portable access to patient data.

Augmented reality (AR)

The ability to surround and overlay digital and virtual objects in the real world is called augmented reality. Information from augmented reality requires low latency and high information processing speed to provide accurate information based on the client's location (Ai et al., 2017). Augmented reality applications are extremely sensitive to latency. A small delay in response time can be detrimental to client capabilities. Therefore, fog computers can become an important player in the field of augmented reality.

Traffic control system

A video camera that detects the ambulance's flashing lights in the traffic control system can automatically change the street lights and open up roads for the vehicle to cross the road. Smart street lights work locally with sensors that detect the presence of pedestrians and cyclists and estimate the distance and speed of approaching vehicles. In addition, smart traffic lights automatically turn on when a sensor detects motion and turn off when vehicles pass by. Smart proximity lights, which work like fog lights, work together to create a green traffic light and send a warning signal to approaching vehicles. A traffic control system can help prevent

accidents, maintain steady-flow traffic, and collect relevant data to evaluate and improve the system's performance(Saurez et al., 2017).

2.4.3. **Fog computing-related paradigms**

The term "fog computing," which refers to the delivery of cloud services to the edge of an enterprise network via Mobile Edge Computing (MEC). Fog computing has the advantage of allowing a single processing device to gather data from multiple sensors and act on it. For example, a smart robotic vacuum cleaner receiving data from multiple sensors installed in a house capable of detecting any dirt and sending any command to the vacuum cleaner. There are several related computing paradigms available, such as edge computing, mobile edge computing (MEC), and mobile cloud computing (MCC).

Edge Computing (EC)

Bringing computational resources closer to the source of data production is a fundamental principle of edge computing (Naha et al., 2018b). End devices and edge devices are components of edge computing. Smart objects, sensors, cell phones, and other such end devices are examples of end devices, while wireless access points, base stations, and bridges are examples of end devices near the data generation site. Instead of providing all models of cloud services, such as platform as a service (PaaS), software as a service (PaaS), and infrastructure as a service (IaaS), edge computing is designed to produce ultra-low latency computing for the end user.

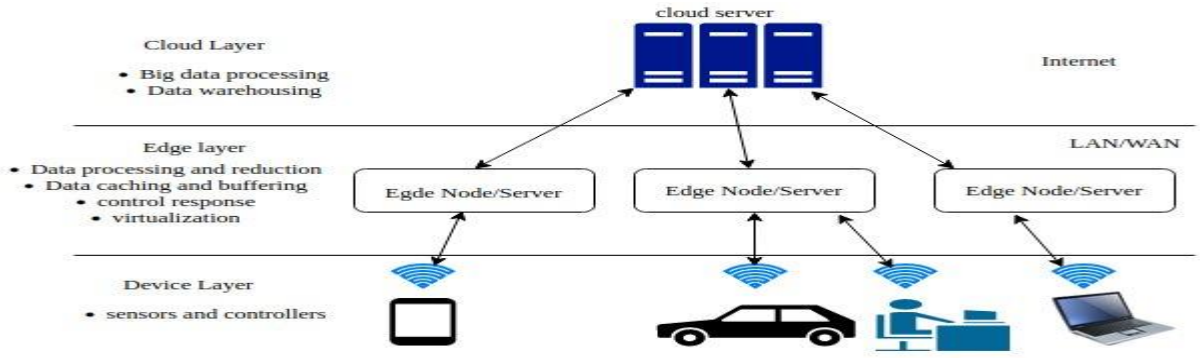

Figure 2. 4 Edge Computing (Steps, n.d.)

Mobile Edge Computing

Mobile edge computing (MEC) creates infrastructure at mobile networks that can support cloud computing and IT services(Mao et al., n.d.). According to (Klas, 2015), MEC evolved from mobile base stations, and as a result of its edge location, MEC provides low latency and real-time services. MEC's basic concept is to execute applications and provide services to cell clients

at base stations(Mao et al., n.d.). The MEC architecture is made up of three basic components: edge devices, edge cloud, and public cloud. Edge devices are IoT and mobile devices that can connect to the network. The edge cloud includes devices that can be deployed at the base station. The edge cloud is in charge of controlling, managing, and sending data from the IoT to the cloud. The public cloud is the third component and contains the infrastructure and services that can be accessed through the Internet.

Mobile cloud Computing

Mobile Cloud Computing (MCC) combines the benefits of mobile computing, cloud computing, and mobile internet in general. The primary goal of cloud computing is to provide isolated virtualized computing, storage, and communication resources that are accessible to end users(Kumar, n.d.). Amazon EC2, Microsoft Azure, Google, and Aneka are a few examples of cloud computing infrastructures and platforms. In a mobile environment, mobile cloud computing enables on-demand access to network, server, application, storage, and computing resources. MCC also focuses on resource management, which should be simple(Kumar, n.d.). Because centralized cloud servers in an MCC infrastructure are located far away from end devices, they are less productive in a computation-intensive environment. For example, while mobile applications are connected to the cloud, they may experience network latency or disconnections.

Table 2. 1 summary of fog computing and related technology

Attributes	Mobile Edge Computing	Edge Computing	Mobile Cloud Computing	Fog Computing
Service type	Less global	Local	Local	Less global
Distance from the user	Close	Very close	Far	Relatively close
Architecture	Localized	Distributed	Central cloud with distributed mobile	decentralized
Latency	Low	Low	Relatively high	Low
Connection to the cloud	Yes or no	Yes or no	Yes	Yes
Operators	Network infrastructure providers	Network infrastructure providers and local business	Users and cloud providers	Users and cloud service providers.

2.4.4. **Fog versus cloud**

Whether latency-sensitive applications can be supported while maintaining sufficient quality of service is a typical example that is frequently used to distinguish between fog and cloud computing (QoS)(Yousefpour, Fung, Nguyen, Kadiyala, Jalali, Niakanlahiji, Kong, Jue, et al., 2019). When compared to standard cloud computing, latency can be significantly decreased by placing fog nodes close to IoT source nodes. Although this illustration provides a clear justification for fog, latency-sensitive applications are merely one of several that do so. In contrast to centralized cloud data centers, nodes in fog computing are typically distributed in less concentrated places. Fog nodes are geographically diverse and widely dispersed. In contrast to the centrally built security procedures in specialized buildings for cloud data centers, security in fog computing must be supplied at the edge or in the specific locations of fog nodes. The decentralized nature of fog computing enables objects to either utilize fog resources as clients of the fog or serve as fog computing nodes themselves (for example, an automobile operates as a fog node for onboard sensors). The size of the hardware components involved with these computing paradigms is largely responsible for the distinctions between cloud and fog computing. While fog computing offers intermediate availability of computer resources at lower power consumption, cloud computing offers high availability of computing resources at comparatively high-power consumption. Large data centers are often used for cloud computing, while small servers, routers, switches, gateways, set-top boxes, or access points are used for fog computing. Hardware for fog computing can be placed closer to consumers because it takes up considerably less area than hardware for cloud computing. Cloud computing can only be accessed through the network core, but fog computing can be accessed by linked devices from the network's edge to its center. Furthermore, continuous Internet access is not required for the fog-based services to function Cloud computing, on the Fog helps devices measure, monitor, process, analyze, and react, and distributes computation, communication, storage, control, and decision making closer to IoT devices(Yousefpour, Fung, Nguyen, Kadiyala, Jalali, Niakanlahiji, Kong, Jue, et al., 2019).

2.5. **Overview of Internet of Things (IoT)**

The Internet of Things (IoT) is a collection of interconnected devices that are connected to a network and/or to one another and exchange data without the need for human-machine interaction. In other terms, the Internet of Things (IoT) is a collection of electronic devices that

can communicate with one another. Smart factories, smart homes, medical monitoring devices, wearable fitness trackers, smart city infrastructures, and vehicle telematics are just a few examples(February et al., 2020).The Internet of Things (IoT) is a self-configuring network that connects real-world things to the Internet which enables communication with other devices. This communication leads to the emergence of new services everywhere(Negash et al., 1999). the structural component of IoT can be defined as an interconnected worldwide network based on sensory, communication, networking, and information technologies using the terms "Internet" and "Things ". The vision of the IoT is to connect a huge number of intelligent things to the internet. Many technologies are used to connect those devices like Wi-Fi, Bluetooth, RFID, and near field communication (NFC). The significance of this technology become more priceless due to the applications and their benefits(Sudheer & Vamsi Krishna, 2019).

2.5.1. **Internet of Things applications**

IoT applications can now be found in a variety of industries, including healthcare, the environment, commerce, and infrastructure supplies (Survey, 2020). IoT has become one of the potential developing technologies that has attracted many academics and practitioners as a new trend as a result of its application in various areas (Ghobaei-Arani et al., 2020).

IoT devices monitor health-care services such as heart rate, patient movement, surgical operations, and other medical readings. In actuality, activities such as the fuel and mineral industries, as well as manufacturing that cannot be completed by humans, are managed by IoT devices, allowing them to prevail in solving many challenging issues(Survey, 2020). Furthermore, certain promising technologies, such as smart homes, smart cities, smart farms, smart automobiles, and others, are the outcome of IoT device engagement (Ghobaei-Arani et al.,2020). They have several uses in the daily lives of individuals, businesses, and society. The Internet of Things application encompasses a smart environment, such as transportation, agriculture, factory, supply chain, healthcare, culture and tourism, environment, and energy (Hafdi et al., 2019). Furthermore, smart vehicles in smart transportations can detect road conditions, distance from other cars, barriers, potential collisions, fire, and other hazards to ensure a safe and automated travel(Hafdi et al., 2019).

2.6. **Task Scheduling in Fog Computing**

Task scheduling is a set of rules and regulations that are used to assign jobs to the appropriate resources (CPU, memory, and bandwidth) in order to achieve the best level of performance and

resource utilization feasible. Due to the variability of fog nodes in processing and storage capacity, it is the most difficult to accurately select which nodes on which tasks can be performed to achieve a better quality of services. The diverse character of fog nodes is the primary challenge in fog computing resource allocation and scheduling (Naha et al., 2018). Computing resource management is defined as the approach for managing computing resources (limited-availability components) (Naha et al., 2018). The availability of adequate task scheduling for effective processing is unavoidable due to the diverse and heterogeneous nature of fog computing resources (J. Wang & Li, 2019).

2.6.1. **Scheduling algorithms**

Scheduling is the process of correctly assigning tasks to resources in order to maximize resource utilization while meeting task requirements. It also refers to how processes are assigned to the available processors for execution. The primary purpose of scheduling is to reduce the time it takes for a parallel application to complete by appropriately and carefully distributing jobs to the right processors(Parikh, 2013).

Task scheduling is the process of creating a queue to run a series of applications over a set period of time. It is the assignment of tasks to a specific set of resources that may be deployed across various administrative domains. The scheduler's principal responsibility is to choose which process to run in the next step from a collection of applicable processes(Parikh, 2013).Some of the task scheduling modes that are used widely are discussed as follows:

Static scheduling algorithms: before starting the scheduling process, details of the tasks must be totally known to the scheduler in static scheduling methods. All tasks will be submitted to the system at the same time, regardless of the state of the system's resources or their accessibility (Alizadeh et al., 2020).

The **First-Come-First-Served (FCFS)** algorithms are well-known traditional examples of these scheduling algorithms and the basic technique used in cloud computing. FCFS simply analyzes task arrival times and ignores all other factors when making scheduling decisions.

This algorithm executes tasks in the order in which they arrive. It is not very efficient in terms of time-sensitive applications. The reason for this is that task wait times are long. Large tasks that require a long execution time and delay-tolerant tasks arrive first, forcing time-sensitive tasks to wait a long time.

Round Robin algorithm: This type of algorithm allows a fixed amount of time for each work, known as quantum, which is then processed using the FCFS method and the round robin scheduling was designed based on the CPU time distribution among the tasks of the schedule. It is unaffected by task duration or priority. If a task's allotted time is up, the processor will move on to the next task in a queue (Aladwani, n.d.).

Min–Min algorithm: This algorithm prioritizes small tasks over large tasks, resulting in long delays for large tasks. The Min-Min algorithm starts with a list of all unmapped tasks with the same minimum completion time. The task with the lowest overall completion time from the set is then chosen and assigned to the corresponding machine(Science & Teshome, 2019).

The **Max-Min algorithm** prioritizes large tasks over small tasks, resulting in long delays for small tasks. The Max-Min Algorithm is very similar to the Min-Min Algorithm. It also starts with an array of all unmapped tasks. The set of minimum completion times is then determined. Now, the task with the highest overall completion time from the set is chosen and assigned to the corresponding machine (hence, Max-Min)(Science & Teshome, 2019).

Dynamic scheduling: It considers the current state of virtual machines, does not require prior knowledge of the system's overall state, and assigns tasks based on the capabilities of all available virtual machines(Endo et al., 2016). At the time of execution, resources are allocated based on the requirements. It is possible to change the resource allocation during execution.

Preemptive scheduling: Each task is interrupted during execution and can be passed to another resource for completion(Endo et al., 2016).

Non-preemptive scheduling: VMs are not reassigned to new tasks until the scheduled task has completed its execution(Endo et al., 2016).

Batch mode scheduling: is a batch of tasks that are collected and given to the scheduler for allocation. It might take some time to provide a decision for processing and get the result. Some algorithms such as the Max-Min algorithm, First-Come-First-Serve algorithm, Short-Job-First algorithm, and meta-heuristic algorithms such as the genetic algorithm and particle swarm optimization can be used in batch processing (Kabirzadeh et al., 2018).

Immediate mode scheduling: when new tasks arrive, they are scheduled to VMs directly without waiting for the next time interval that might be either when the scheduler gets activated (Stream, n.d.).

2.6.2. **Nature-Inspired Metaheuristics scheduling**

Each meta-heuristic algorithm makes a trade-off between randomization and local search. It should be noted that there is no single definition of heuristics and metaheuristics in the literature; some authors use the terms "heuristics" and "metaheuristics" interchangeably (Liu et al., 2017). However, recently, there has been a tendency to refer to all stochastic algorithms involving randomization and local search as "metaheuristics." This concept is also applied here. Randomization is an efficient method for moving from local to global search. As a result, almost all metaheuristic algorithms can be used for global optimization.

Exploration and exploitation are the two most important parts of any metaheuristic algorithm. Exploitation involves searching the entire search domain and obtaining different solutions, whereas exploitation involves finding the best available solutions in any domain, applying knowledge about the selected solution, focusing the search (local search) in that domain, and selecting the best candidate solutions(Carbas et al., n.d.). Although randomization is useful for solutions in the exploration phase, it prevents the local optimum from being hit. The exploration stage allows the algorithm to search the solution domain more efficiently. In other words, the search area is explored to find the best solution; the neighborhood of that solution is searched; meanwhile, the search continues to find the best solutions(Carbas et al., n.d.).

There are many metaheuristics algorithms such as grey wolf optimization algorithm, particle swarm optimization algorithm, genetic algorithm and etc.

2.6.2.1. **Grey Wolf Optimization (GWO)Algorithm**

Grey wolf is a population-based meta heuristic algorithm. Compared to other popular population-based techniques like the genetic algorithm (GA), particle swarm optimization (PSO), firefly algorithm (FA), artificial fish swarm algorithm (AFSA), and differential evolutions algorithms (DE), it has higher convergence properties. grey wolf optimizer algorithms are simple to operate in the optimization process, easy to implement, and has few adjustment parameters(Li-wenGuo, 2019).

Being at the top of the food chain, grey wolves are regarded as apex predators. The majority of grey wolves prefer to live in packs. Average group size is between 5 and 12. They have a fairly rigid social dominant hierarchy, as depicted in Fig. 2.5, which is very interesting. Alphas, or the leaders, are both male and female. The alpha is mostly in charge of making decisions on hunting, where to sleep, when to wake up, and other issues. The pack must follow the alpha's directives. However, an alpha has also been seen to behave democratically by adhering to the pack's other

wolves. The entire pack bows to the alpha during gatherings by holding their tails down. Since the pack should obey the alpha wolf's commands, the alpha wolf is also referred to as the dominating wolf (Mirjalili et al., 2014). Only the alpha wolves are permitted to mate. It's interesting to note that the alpha is not always the most physically powerful member of the pack, but rather the greatest at leading the pack. This demonstrates that a pack's structure and discipline are far more crucial than its physical power.

Beta is the second position in the grey wolf hierarchy. The betas are the wolves below the alpha who help alpha in making decisions or participating in other pack activities. The beta wolf, which can be either male or female, is most likely the best choice to take over as alpha in the event that one of the other wolves passes away or gets too old. The beta wolf not only commands the other subordinate wolves but also ought to revere the alpha. It serves as the alpha's counsellor and the pack's enforcer of discipline. The beta provides feedback to the alpha while also reinforcing the alpha's orders throughout the pack.

Omega is the grey wolf with the lowest ranking. The omega serves as the scapegoat. All other dominant wolves must always yield to omega wolves. They are the final wolves that are permitted to consume food. Although it could appear that the omega is a relatively unimportant member of the pack, it has been noted that when the omega is lost, the entire pack has internal conflict and issues. This is a result of the omega's violent and irate wolves' outbursts. This helps to preserve the dominance structure while also appeasing the entire pack. Sometimes the pack's babysitters double as the omega.

A wolf is considered subordinate if it is not an alpha, beta, or omega (or delta in some references). Alpha and beta wolves must subject to delta wolves, but they rule the omega. These people include elders, scouts, sentinels, hunters, and caregivers. Scouts are in charge of keeping an eye on the perimeter of the area and alerting the pack to any danger. Sentinels watch over and ensure the safety of the pack. Wolves with experience who were once alpha or beta are known as elders. When hunting animals and supplying the pack with food, hunters assist the alphas and betas. The final duty of the caretakers is to tend to the sick, injured, and frail wolves in the pack. According to(Mirjalili et al., 2014) the main phases of grey wolf hunting are as follows:

- ✓ Tracking, chasing, and approaching the prey.
- ✓ Pursuing, encircling, and harassing the prey until it stops moving.

- ✓ Attack towards the prey.

the social hierarchy of grey wolves

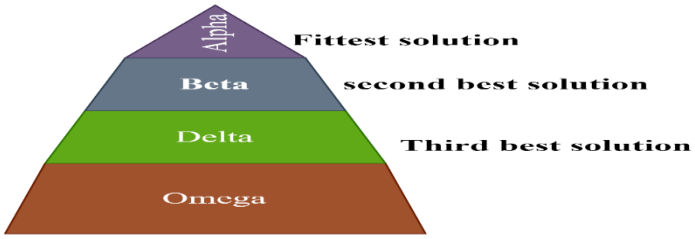

Figure 2. 5 social hierarchy of grey wolves

2.6.2.2. **Particle Swarm Optimization Algorithm**

Particle Swarm Optimization is a random optimization technique (Juneja & Nagar, 2016). In the PSO algorithm, a group of particles in the search space work together to optimize a fitness function, much like a flock of birds might when foraging for food in the wild. The particles evaluate their quality or fitness at a given position in the search space after being distributed at random. Each particle then shifts to a new position that provides a higher fitness than the prior position for a predetermined number of iterations. Based on the history of each particle's best past and present locations in comparison to the best positions reached by other particles in the swarm, plus some random disturbances, this movement is determined(D. Wang et al., 2018).

With a set number of particles cooperating, the swarm therefore finds in following iterations the fitness function solution that is most optimal in the problem space. Depending on the algorithm's intended use, the fitness or objective function in PSO is a performance evaluation criterion. A mathematical framework is typically used to describe the performance criterion in order to quantify the system performance attained through a performance index.

The n particles that make up the fundamental particle swarm optimization technique each represent a potential solution to the fitness function in the D-dimensional search space.

Three factors have an impact on the particle's status.

- ✓ Its own inertia.
- ✓ Personal most optimal position.
- ✓ Swarm’s most optimal position.

2.6.2.3. **Improved hybrid grey wolf optimization algorithm**

Chaos initialization

GWO algorithm usually solves function optimization problem by using randomly generated data as the initial population information, which will not retain the diversity of the population and lead to the poor optimization result of the algorithm. Therefore, this paper proposes Tent chaotic map to initialize the population. Chaotic motion has the characteristics of randomness, regularity, and ergodicity. When solving function optimization problems, these features can lead algorithm to escape local optima, so as to maintain the diversity of the population and improve the global search ability. Chaotic maps include Tent maps, Logistic maps and so on. However, different chaotic mappings have different search characteristics. So far logistic map is mostly used in the literature. But it has a higher value rate in [0, 0.1] and [0.9, 1] that leads to inhomogeneous distribution of values.) proves that the Tent map can perform better than the Logistic mapping in traversal homogeneity and generate a more uniform initial value between [0, 1], so as to improve the optimization speed of the algorithm. Therefore, this paper proposes Tent chaos initialization, that is, we use Tent chaos to initialize the grey wolves.

Nonlinear control parameter strategy

The GWO algorithm is mainly composed of two steps: the prey positioning and the grey wolf individual's predatory behaviour. the parameter A plays a very crucial role in balancing the global exploration and local exploitation capability of the GWO algorithm. When $|A| > 1$, the group will expand the search range to find a better candidate solution, which is the global exploration ability of the GWO algorithm. When $|A| < 1$, the group will narrow the search range and perform detailed search in the local area, which is the local exploitation capability of the GWO algorithm. At the same time that in the process of iteration, the value of A changes continuously with the change of control parameter a. the control parameter a decrease linearly with the increase in the number of iterations. However, the optimization process of GWO algorithm is very complicated. The linear change of parameter a cannot reflect the actual optimization search process of the algorithm. the control parameter a varies nonlinearly with the number of iterations. And through the standard test function optimization results show that the use of nonlinear change strategy is better than the linear strategy optimization. But they still cannot meet the needs of the algorithm.

PSO thought

In the process of location updating, GWO algorithm takes into account only the location information of individual wolves and the optimal solution, second-best solution, and third-best solution location information of the wolf pack, which realizes the exchange of information between individuals and wolf pack. But it ignores the exchange of information between the wolf and its own experience. Therefore, the idea of PSO algorithm is introduced to improve the location updating process. In the PSO algorithm, the current position of the particle is updated by using the best position information of the particle itself and the best position information of the group. This study combining with PSO algorithm will introduce the optimal location of individual experience into the position updating formula, which enables it to keep its own optimal position information.

2.7. Experimentation Tools

Cloud computing and fog computing simulation tools are virtual environments that are designed and developed to resemble physical environment aspects in order to reduce the expense of purchasing and configuring physical devices for developing and simulating research work. Cloudsim, Cloud Analyst, and iFogSim are the most frequently used simulators.

Cloudsim

Is an open-source java-based simulation tool with multiple layers for modeling, programming, and cloud-based simulation experiments? The GRID laboratory at Melbourne University created and developed it(Calheiros et al., 2011). Cloudsim offers a variety of policies for virtual machine allocation, network connectivity within elements, processor core flexibility, and cloudlet management(Suryateja, 2016). The provision of computing resources, virtual machine configuration, data center administration, and user task management are all managed by classes in cloudsim. Researchers can use this simulator to create and develop cloud-related solutions without wasting time and money configuring physical devices.

The main advantages of using Cloudsim for initial performance testing are (i) time efficiency: implementing a test environment for cloud application delivery requires very little effort and time and (ii) flexibility and ease of use: developers can model and test the performance of their application services in heterogeneous cloud environments (Amazon EC2, Microsoft Azure) with few programming and deployment requirements(Calheiros et al., 2011).

In other words, it provides cloud computing environment modeling and simulation capabilities. Cloudsim claims that the user attempts to submit his requests in the form of cloudlets. Each cloudlet has its own file size, number of instructions to be executed, and so on. These cloudlets will be submitted to the broker to be scheduled onto VMs. The ability to create broker-driven policies gives Cloudsim an advantage. In cloudsim, the defined class, VM, represents the virtual machine that can be created on the hosts. The broker is responsible for creating hosts and assigning each VM to a different host.

iFogsim

iFogSim is an evaluation framework that allows for the repeatable measurement of the performance of resource management rules on an IoT or Fog computing infrastructure to model an IoT or Fog environment(Gupta & Buyya, 2017). iFogSim was developed as an extension of Cloudsim, and thus inherited a number of Cloudsim capabilities. The simulator is designed for the Fog computing environment. As a result, it is more suited for resource management and allocation approaches. It can measure latency, network congestion, energy use, and cost(Gupta & Buyya, 2017). Moreover, iFogSim provides an environment for the development and evaluation of resource management and different scheduling policy between IoT devices and fog devices. However, it has a limitation in clearly identifying horizontal communication facilities among fog nodes and provision of migration strategies(Gupta & Buyya, 2017).

Cloud Analysts

Cloud Analyst is a simulation tool that is built on top of the Cloudsim tool set and is used to model large scale Internet and Internet application behaviors in cloud and fog computing environments(Calheiros et al., 2011). The java programming language is used to create it. As a development environment, you can use any Java IDE, such as NetBeans or Eclipse. It adds additional extensions to the Cloudsim toolkit.

Application users: Cloudlets are used as traffic generators with user-configurable behaviors. **Internet**: network delays are used to define the transfer of tasks from IoT devices to fog nodes and cloud servers. Bandwidth constraints are used to model the uplink and downlink data transmission rates.

Service Brokers: This component manages a data center's CPU and virtual machines.

graphical user interface (GUI): The Cloud Analyst simulator has a graphical user interface (GUI) for setting simulation input and behavior. It is used to display experiment results and to allow users to repeat tests.

Saving simulations and results: The simulator has a feature that allows you to store simulations and results as pdf files.

Region: This component allows you to assess the proximity and distance between fog nodes and data centers.

Cloudsim was chosen for modeling of the suggested algorithm for this work because of its very extensive capabilities to help the assessment of cost and delay on fog-cloud environments. Furthermore, our approach necessitates the assignment of the task by both fog and cloud nodes. Cloudsim simulator was chosen for this purpose since it supports both computing environments.

2.8. Related Works

In recent years, many researchers have contributed to task scheduling, resource management and load balancing algorithms for both fog and cloud computing. They are NP-Complete problems, so there is still room for improvement.

(Hoang & Dang, 2017).proposed a method for minimizing the make-span of tasks in a fog-cloud environment. They proposed a region divided into two sections: fog devices as areas that address latency sensitive tasks and cloud servers to address compute-intensive tasks. However, they didn't consider cost reduction at both fog and clouds. (Properly they had considered only time).

(Kabirzadeh et al., 2018).hyper-heuristic algorithm is introduced which is a combination of particle swarm optimization (PSO), GA, ACO and SA for resource utilization at fog nodes. The simulation was conducted by an iFogSim simulator and they found reduction in network bandwidth and execution time by using the proposed algorithm. Moreover, energy consumption was found to be less than that of PSO, ACO, and SA though it became the same with GA. They didn't consider latency as well as the deadlines of the task.

(J. Wang & Li, 2019).proposed methodologies of task scheduling for fog computing based on hybrid heuristic (HH) algorithm which may be a combination of improved PSO algorithm and improved ACO algorithm. The proposed algorithm has appeared to lower delay and energy utilization than ACO and PSO. MATLAB was taken as a test system environment.

(Nguyen et al., 2019).discussed time-cost aware scheduling (TCaS) that pointed at decreasing time of execution and operation cost in a fog-cloud environment. TCaS could be a genetic

algorithm-based and simulated on the iFogSim environment. The studied algorithm has been compared with BLA, Improved PSO and RR planning creating superior results than the three specified algorithms. In any case, they didn't consider deadlines and transmission cost at both fog and cloud environment.

(Alzaqebah et al., 2019).Scheduling based on MGWO in a cloud computing environment. The main objective of this strategy is to decrease both cost and make span. The proposed strategy (MGWO) has significantly faster execution time than both the traditional grey wolf optimization algorithm (GWO) and whale optimization algorithms (WOA) with fitness based on make span, according to the simulation results. However, they did not consider deadlines or resource utilization as working in a cloud environment.

(Bitam et al., 2018). addressed job scheduling using BLA (bees life algorithm) at fog devices. They attempted to get an optimal solution for the trade-off between execution time of CPU and allocated memory needed to complete the services. They improved time of execution and cost of execution at fog nodes. Nevertheless, latency, deadlines, resource utilization has not been considered. In addition to that the objective is only for fog environments which are not cloud supported.

In this study (Hosseinioun et al., 2020)(Hosseinioun et al., 2020).an energy-aware strategy was introduced using a dynamic voltage and frequency scaling (DVFS) approach to reduce power use. In addition, a hybrid invasive weed optimization and Culture (IWO-CA) algorithm was connected in order to develop efficient allocation clusters. Testing found that the algorithms studied improved on various current algorithms in terms of energy use.

(Alsadie & Member, 2021).This study proposes the TSMGWO, a metaheuristic grey wolf optimizer-based approach for finding an optimal or near-optimal solution to the task scheduling problem by simultaneously considering multiple conflicting objectives such as make-span, resource utilization, degree of imbalance, and throughput.

(Nikoui et al., 2020). In this study, we propose a cost-aware genetic-based (CAG) task scheduling algorithm for fog-cloud environments, which improves the cost efficiency in real-time applications with hard deadlines. The performance results show that the proposed algorithm provides better efficiency in terms of the cost and throughput compared to Round-Robin and Minimum Response Time algorithms.

(Ababneh, 2021). A Hybrid Approach Based on Grey Wolf and Whale Optimization Algorithms for Solving Cloud Task Scheduling Problem. To address the problem of task scheduling in cloud computing, which necessitates nontraditional optimization attitudes to achieve the problem's optimality, the current paper proposes a hybrid multiple-objective approach called hybrid grey wolf and whale optimization (HGWWO)algorithms, which integrates two algorithms, namely, the grey wolf optimizer (GWO) and the whale optimization algorithm (WOA), with the goal of combining the advantages of each algorithm for minimizability. According to the results of the experimental work, the proposed approach has the capability of outperforming the original algorithms GWO and WOA on their own in terms of costs, energy consumption, makespan, resource use, and degree of imbalance. The deadline was not considered and was only conducted on cloud computing.

(Science & Teshome, 2019). Design and development of task Scheduling algorithm for fog Computing environment. This proposed algorithm introduces approach to optimize task scheduling problem in terms of response time. In the algorithm, tasks could be categorized as simple, medium and large or complex. This algorithm assigns simple and medium tasks to the nearest fog layer using their priority and large tasks to aggregate fog layers. To show the performance of the proposed algorithm a comparison was made with FCFS and SJF. The experimental results show that the proposed algorithm has better performance on reducing the response time of tasks when compared to FCFS and SJF. Cost and deadline of tasks were not indicated.

The following table shows a summary of each literature of review including their strength, weakness and ideas for which algorithm developed.

Table 2. 2 summary of literature review and related work

Authors and published year	***Main ideas***	***limitation***	***Finding***	***metrics***
(Hosseinioun et al., 2020)	A new energy-aware tasks scheduling approach in fog computing using hybrid meta-heuristic algorithm	In the study latency and cost did not indicate	When compared to other algorithms, their proposed algorithm used less energy as demonstrated by the results of their simulation.	Reducing energy consumptions
(Hoang & Dang, 2017)	a heuristic-based algorithm task scheduling fog-cloud computing	Cost and delay were not taken into account	They proposed a region that is divided into two areas that is a fog device as the area that addresses latency-sensitive tasks and a cloud server to address compute-intensive tasks.	Completion time, resource utilization.
(Nguyen et al., 2019)	Evolutionary Algorithms to Optimize Task Scheduling Problem for the IoT Based Bag-of-Tasks Application in Cloud–Fog Computing Environment	They failed to consider the effect deadlines, in both the fog and cloud environments	The proposed algorithm has been compared with BLA, modified PSO, and RR scheduling producing better results than the three mentioned algorithms.	execution time, operation cost

(J. Wang & Li, 2019)	Task Scheduling Based on a Hybrid Heuristic Algorithm for Smart Production Line with Fog Computing	Cost, and deadline were not considered	The proposed algorithm has shown lower delay and energy consumption than ACO and PSO.	Reduce delay and energy consumption
(Alzaqebah et al., 2019)	MGWO algorithm task scheduling on cloud computing	They only focused on cloud computing and didn't take deadlines into account	MGWO has better performance in terms of make span, cost, and degree of imbalance than traditional (GWO) and (WOA) algorithms.	Make span, cost, degree of imbalance
(Bitam et al., 2018)	Fog computing job scheduling optimization based on bee's swarm.	cost, and completion time were didn't considered	They improved the time of execution and cost of execution fog nodes. As their experiment shows, they outperformed PSO and ACO algorithms in terms of execution time and execution costs.	Resource utilization, Latency
(Alsadie & Member, 2021)	Optimizing Task Schedule Using Multi-Objectives Grey Wolf Optimizer for Cloud Data Centers	They did not consider the cost and the deadline	The results are compared to heuristic FCFS as well as meta-heuristic methods PSO, GA, and WOA. In comparison to those algorithms, the TSMGWO approach outperformed.	make span, degree imbalance, resource utilization, throughput

(Ababneh, 2021)	A Hybrid Approach Based on Grey Wolf and Whale Optimization Algorithms for Solving Cloud Task Scheduling Problem	The deadline was not considered and was only conducted on cloud computing.	the proposed approach has the capability of performing at a superior level by comparison to the original algorithms GWO and WOA on their own with regard to costs, energy consumption, makespan, use of resources, and degree of imbalance.	minimizing costs, energy consumption, and total execution time
(Science & Teshome, 2019)	An Improved Grey Wolf Optimization Algorithm Based Task Scheduling in Cloud Computing Environment	latency was not properly indicated	The superiority of proposed technique is evident from the simulation results that show a comprehensive reduction in task completion time and cost	processing time and cost

As indicated in the table, no approach was designed to simultaneously implement all the objectives of quality of service (QoS) which includes latency minimization, cost reduction, deadline and resource utilization. Many designed an environment that will be suitable to enhance one or two elements of QoS (Quality of Service) at the expense of other elements. As an example, Wang J, Li D,(J. Wang & Li, 2019). An algorithm was proposed that improved energy reduction and delay minimization, but cost and deadlines were not considered while improving energy reduction and latency minimization. Essentially, others also attempted to improve some components of QoS with no optimizing the trade-off between other QoS components. The proposed approach has been designed to minimize cost and latency at fog nodes taking deadline and resource utilization into consideration so that the needs of clients will be met.

CHAPTER THREE

3.Designing and Implementation

This chapter discusses designing proposed solutions, system designing consideration, algorithm system model, and implementation of the proposed solution.

3.1. Designing Proposed Solution

As we have already indicated, this study provides a detailed discussion of the cost-latency optimization task scheduling algorithm's latest and modified architecture, policy, design considerations, and parameters. The use of the improved hybrid grey wolf optimization algorithm to build a task scheduling design that fits the task's deadline while saving costs and mapping to virtual machines is extensively discussed. Allocating tasks to appropriate accessible nodes improves the quality of services such as latency, cost, and deadline.

3.1.1. Fog-Cloud Computing Architecture Adapted

Since there is no standard fog computing architecture accessible to date (Naha et al., 2018), it is critical to choose and adapted an existing architecture that best fits the proposed algorithm. To achieve the aims of the proposed algorithm, so for this research, the four-tier architecture has been selected. It is more suitable for providing IoT applications with different interests. For example, applications that need more processing power can use the aggregate(upper-level) layer while others can use the low-level fog layer. The low-level fog layer used for quick feedback and the aggregate one is used for medium scale and medium-term analysis. The components in the four-layer are the IoT devices in the first layer, Lower-level fog nodes in the second layer, Aggregate(upper-level) Fog nodes and Cloud servers in the third and fourth layer respectively.

IoT devices: This layer contains data sources such as Wi-Fi sensors, actuators, and smart devices that generate various types of data. This layer is responsible for generating data from the environment and sending it to the upper layers such as low-level fog nodes, aggregate fog nodes, and cloud nodes for processing.

Lower-Level (Nearest) Fog Layer: The IoT layer delivers data to the nearest low-level fog node for temporary storage or computation. It is composed of IT-enabled edge devices such as smart base stations, switches, gateways, and routers. The fog broker takes the required data from the first layer and assigns it to accessible fog nodes if they can process it; otherwise, the data is

delivered to compute-intensive fog nodes such aggregated fog nodes and cloud nodes. This second fog layer, for example, can be used to investigate motion detection in surveillance cameras(Byers, 2017).

Aggregate(upper-level) Fog Node Layer: This layer consists of an aggregate fog node which is much more powerful in terms of computing power and storage capacity than the second tier. It is composed of a collection of micro data centers, fog servers, smart gateways, and smart routers. Those devices have more computation capability and storage capacity when compared to lower-level fog nodes. Tasks that have a latency-tolerant(less-sensitive) deadline would be executed in this layer. It provides a more detailed and intensive analysis such as surveillance camera video pattern matching (Byers, 2017).

Cloud Layer: This layer contains cloud servers and data centers that provide network, computing, and storage. The cloud layer, which incorporates devices with large computational capabilities, massive storage, and big data processing, is the fourth tier. Tasks having extremely high deadlines (years, months, weeks, days, and hours) from a large-scale ecosystem would be completed at this tier. Long-term pattern matching, long-term natural disaster prediction, climate change prediction, and weather forecasting, for example, all necessitate the use of a cloud layer. The task is sent to this layer by the aggregate fog layer for storage and further processing. This layer does more advanced computer processing and maintains data indefinitely, either as a backup or to provide users with permanent access(Sharma & Saini, 2019).

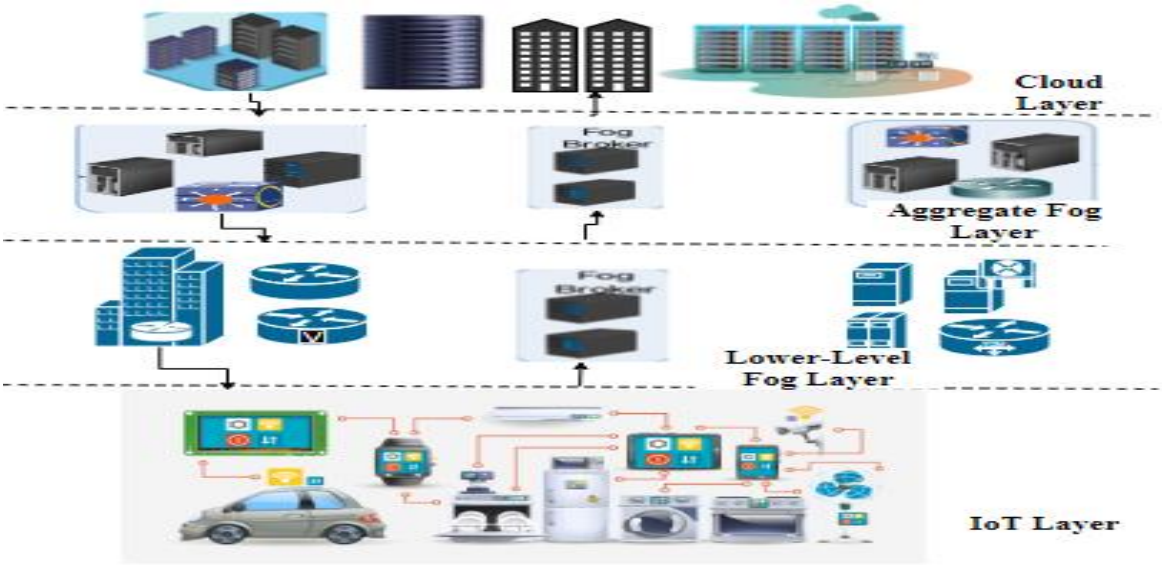

Figure 3. 1 Fog-Cloud Computing Architecture Adapted

3.1.2. System Design Consideration

Cost Reduction

The cost is the price paid to perform the task on the selected nodes, which is related to the node capacity as well as the price associated with its usage. The cost of transferring a task from one node to another can be estimated in terms of the time it takes the processor to complete the task, the bandwidth required, and the cost of memory usage.

Latency reduction

Latency is the amount of time it takes from submitting a task to the user completing the task. It is calculated by adding the task's waiting time, transmission delay, propagation delay, and processing delay. It denotes minimizing execution time and thus meeting the task's execution constraints. In terms of execution time, a good task schedule has a lower value. It is typically calculated as the time it takes a task to complete its execution across multiple virtual machines.

Response time: This is the amount of time between when a task is submitted and when it is finished. It can also be calculated as the total of the waiting time and the execution time.

Waiting time: is the period of time between task submission and task execution. It also includes the period of time a task takes to acquire additional resources or wait for certain events. For example, if a low-level fog node is unable to complete a task, it should be transferred to an upper-level fog node and cloud nodes if the work's deadline has not yet passed.

Deadline

The ultimate goal of fog computing deployment is to reduce latency by using computing devices at the near edge of the network. Although latency can be reduced, meeting deadlines remains a challenge due to the low computing power of the fog nodes. As a result, we need to explore additional procedures to ensure that the quality of service provided meets the requirements of the assignment. Quality of service can be achieved by implementing task scheduling algorithms that schedule tasks generated by IoT devices based on criteria such as deadline, task size, and task type.

Data size

Although fog computing was introduced to reduce task response time by executing at the network's edge, this is not always true due to task differences. Tasks have varying deadlines, so a request should be serviced based on its nature. This is due to the fact that fog nodes at the network's edge have limited computation power and storage capacity, which must be carefully

managed in order to get the most out of this paradigm in meeting task requirements. Tasks must be assigned and processed in fog nodes with reasonable estimated response and service times. To make this work, separating the types of tasks is a good idea. Some tasks may take longer to complete and require more computing power, whereas others may not require as much processing power but are latency sensitive. Other tasks may have deadlines that the receiving node cannot meet. Furthermore, the size of tasks generated by IoT devices determines the connection technology and standard used by the device to communicate with fog nodes. For example, IoT sensors generate a specific type of task. To communicate with the fog node, these units should use IEEE 802.15.4, ZigBee or NB-IoT technologies.

In order to calculate data transfer and network latency, identifying the kind of incoming activities is a crucial consideration. We classify and choose the places where the activities are executed based on these task-specific characteristics. The three types of IoT tasks small, medium, and large are categorized in this study based on the needed resource capacity(Byers, 2017). Small activities require little computational power, so they are finished at the closest (low-level) fog node according to deadlines. Medium and large tasks, on the other hand, need a lot of computational power, so they must be finished at aggregated (upper-level) fog and cloud nodes, respectively(Byers, 2017).

Table 3 1 The characteristics of tasks based on the size of their data

Small Task		**Medium Task**		**Large Task**	
Minimum	Maximum	Minimum	Maximum	Minimum	Maximum
0.1bytes	100KB	100.1KB	200KB	200.1KB	500MB

3.1.3. **Main Components of the System Model**

Various IoT Device Tasks

IoT devices produce various tasks and send those requests to the nearest fog node. The suggested approach routes incoming tasks to the relevant fog nodes based on task type, task size, and task deadline.

Task allocator

All requests coming from endpoints or mobile users are immediately forwarded to a task allocator, who is in charge of evaluating, estimating, and categorizing all tasks that need to be

finished in the Cloud-Fog system. Because the task allocator is close to the Fog nodes, the time required for data transfer between them is low. This is in responsible of classifying and allocating incoming tasks. The decisions are affected by the task size, type, and deadline of task.

Node Controller

This component's primary function is to store all relevant fog node information such as waiting time and queue size. When tasks arrive and are completed at each node, the node controller must update its information on the node. This component assists the fog node server's scheduler component in understanding the node's current condition when assigning tasks to nodes.

Scheduler

Based on the type of task, this layer uses an improved hybrid grey wolf optimizer algorithm to schedule incoming tasks. In order to reduce latency and costs while meeting the deadline, an improved hybrid grey wolf optimizer algorithm is used to allocate the proper low-level fog node, upper-level fog node, and cloud node resources. This algorithm is then optimized using the scalarization method of multi-objective optimization techniques. In our case, latency, and cost were reduced to a single objective using the multi-objective optimization scalarization method. The fitness function of the hybrid algorithm, which is formed using the scalarization method in this work, will evaluate the possible solutions and select the best solution based on the fitness function value. The fitness (objective) function obtained by combining latency and cost objective in equation (Eq 3. 36).

When IoT devices send data to the low-level fog node, task allocator classifies the task and determines the latency sensitivity and task size of the task to assign on low-level fog nodes, upper-level fog nodes, or cloud nodes, as shown in figure 3.2. After determining the latency sensitivity of a task, it can be queued in latency-sensitive, latency-tolerant(less-sensitive), or latency insensitive queues based on its deadline requirement.

Queue

Based on the category of task classifier, there are three queues: one for latency-sensitive and small size tasks, one for latency-tolerant(less-sensitive) and medium task size task, and one for latency-insensitive and large tasks. Three queues are used in this model to hold the task until it is executed. Incoming tasks are queued until they are processed.

3.2. Proposed cost-latency optimization task scheduling Algorithm System Model

The following describes how the proposed cost-latency optimization task scheduling algorithm works. The first tasks generated by various IoT devices are sent to low-level fog nodes (the nearest fog nodes) for computing. Because tasks generated by IoT devices vary in nature (time-sensitivity, size, and type), and the fog node resource is restricted in compute power and storage, all incoming tasks may not be served by the nearest fog node. As a result, the tasks will be evaluated in order to determine and select the best location for execution. The first criterion for scheduling tasks in our system architecture is a task deadline time. All tasks generated by IoT devices will have an acceptable response time, which will be used as a deadline. Another criterion for scheduling tasks is the task size. Data transfer and network delay will be calculated based on the size of task. This outcome is used to estimate the expected service time. So, by combining the two policies, the when and where policy, users can obtain a high level of service with a short total response time. Tasks can be carried out at any of the three layers. Tasks that cannot be completed at the second layer can be completed at the aggregate or cloud layers. When the upper-level fog layer receives a request from a lower-level fog node, it stores or executes data and when the cloud node layer receives a request from a lower-level fog node, it stores or executes.

3.2.1. Proposed solution process

The proposed solution is prepared up of a nature-inspired hybridized grey wolf optimizer search algorithm. In designing task scheduling over a fog-cloud environment, this approach ensures cost reduction, latency reduction, and deadline meeting. To optimize latency and cost, the scalarizing method of multi-objective optimization techniques is used, which states that adding all objectives with some ratio for each objective is used. As a result, a single objective emerges as the best solution to all objectives. As a result, obtaining and evaluating a better solution for a single objective would be simple. Before scheduling, a provider determines the cost and latency ratios that must be considered. Based on this, the scheduler finds the best solution out of all possible solutions by multiplying both latency and cost by the ratio provided and adding the result to compare the solution to other solutions found. Incoming tasks are classified as latency-sensitive, latency-less sensitive, or latency-insensitive, as well as small, medium, or large in

size, with latency-sensitive and small size tasks assigned to the nearest (low-level) fog nodes, latency-less sensitive and medium size tasks assigned to the aggregate (upper-level) of nodes, and latency- insensitive and huge size tasks assigned to cloud nodes.

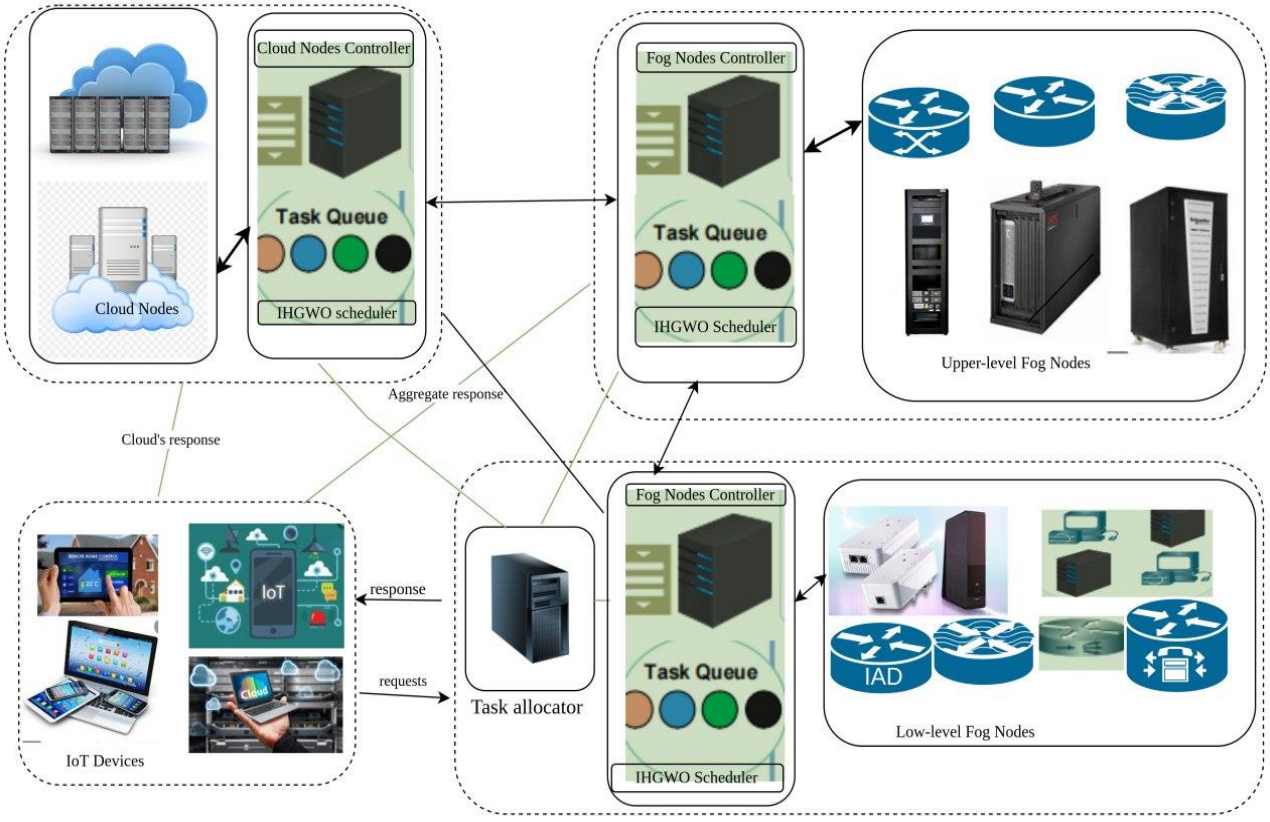

Figure 3. 2 System architecture of the proposed algorithm

3.3. **The Proposed Cost and Latency Optimization Task Scheduling Algorithm**

When requests from IoT applications are sent to the Fog layer and Cloud layer, tasks are assigned to one of three nodes based on their deadlines and task size: nearest fog node, aggregate fog node, and cloud fog node. Task attributes include task size, deadline, input file size, and output file size. Furthermore, they must be executed on the appropriate nodes so that a given deadline is met while maintaining the efficient quality of services such as cost and latency. Because tasks generated by IoT devices vary in nature (latency sensitivity, length, and type), and the fog node's computation resource is insufficient to meet the deadline, some incoming tasks may not be executed in the nearest fog nodes. A deadline is essential in the proposed model

for determining appropriate resource allocation for a given task. The size of the task can sometimes be a determinant of where a task must be executed because a task with a length greater than the capacity of the nearest fog nodes requires upper-level(aggregate) fog nodes to meet their deadline. As shown in Figure:3.2, there are three places for task execution. The first is the nearest fog node, which is designed to perform time-sensitive tasks and has small task size. The second is the upper-level(aggregate) fog node, where latency-fewer sensitive tasks can be executed to meet the deadline as well as medium task size. Latency-insensitive tasks, on the other hand, will be executed on cloud nodes, which is the third place because there is no deadline. As a result, the task classifier component will evaluate tasks in order to place them in appropriate nodes. The task classifier component considers task deadline, task size, and service delay to classify tasks as latency-sensitive, latency-fewer sensitive, or latency-insensitive.

3.3.1. **Task Allocator Model**

As the number of IoT devices increases, it generates a wide range of task requests, from minor to major. This study's improved hybrid grey wolf optimizer-based scheduling algorithm starts with a task classification process based on deadline and task size. Then, using this task classification information, we assign the task to an appropriate node in the hierarchical layers of the fog computing architecture to reduce response time, overall latency, and costs. We divided Internet of Things tasks into three categories: small, medium, and large, as well as latency-sensitive, latency-fewer sensitive, and latency-insensitive. The threshold value is the most important factor in task classification. Tasks generated by IoT devices are typically small in size and require little resource capacity. By definition, fog nodes have low-capacity resources. This variable is used to categorize tasks as small, medium, or large, as well as to order requests in the system based on their Task size.

A small task: is one that does not require a more powerful resource to be processed and is also short in task size and execution time. Because of the low resource capability and short task size, this type of task should be processed at the nearest(low-level) fog node. A task with a short task length necessitates a resource with a low capacity.

Medium tasks: are longer in size than small tasks and should be assigned to aggregate(upper-level) fog nodes by task allocator as well as by checking the deadline threshold values. The task of having a medium task has an average execution time and an average response time, as well as a medium task size.

Large task: is the longest task and should be sent to the cloud layer. The complex task would be processed in the cloud data center because it has greater processing power, storage capacity, and a more reliable server. Because of the task's powerful resources, it also has a fast response time.

This model determines which queue type is assigned to which task in the proposed algorithm. Latency-sensitive and small task size queue, latency-tolerant(less-sensitive) medium task size queue, and latency-insensitive large task size queue are the three queue types. When a task is sent from an IoT device, the proposed algorithm will classify it as either latency-sensitive and small task size, latency-tolerant(less-sensitive) medium task size, or latency-insensitive large task size.

The deadline and size of a task are used to categories where a task should be executed based on a given threshold. Tasks with deadlines ranging from 0.1 microseconds to 100 microseconds, as well as small tasks ranging from 0.1 bytes to 100 kilobytes, must be executed at the nearest(low-level) fog layer for this work, according to(Byers, 2017). Aggregate(upper-level) fog nodes must complete tasks with a deadline of 100microseconds to seconds and a medium size task with an average of 200 kilobytes. Tasks with deadlines ranging from seconds or minutes to days, months, and larger than 1 megabyte in size must also be executed at the cloud layer(Science & Tsega, 2020). The diagram below depicts task classification based on deadline and task size.

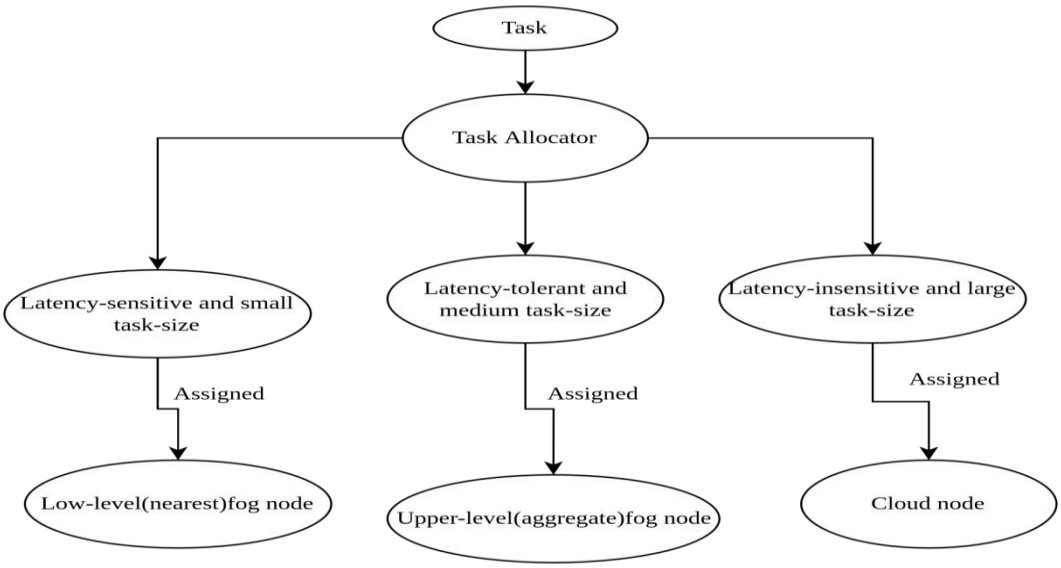

Figure 3. 3 Task Allocators Model

3.3.2. **Node Choice**

The resource allocation process is dependent on queue waiting time, task deadline, transmission delay, propagation delay, and execution delay to make the right decision to assign tasks to specific nodes, as discussed below:

Propagation Delay(Pd_{pro}): is the amount of time it takes for a bit of data to travel across a transmission link that is limited by the speed of light because nothing can travel faster than the speed of light (Khanvilkar et al., n.d.). The amount of time it takes for the first bit to travel over a link between the sender and receiver nodes is referred to as propagation delay. The delay in data propagation is not depend on data size (Khanvilkar et al., n.d.). The propagation delay between IoT and fog is assumed to be˜2μ seconds in this work, while the propagation delay between fog and cloud is assumed to be˜0.8 seconds.

Execution Delay (ED): The amount of time it takes for a task to be finished on a node to which it has been assigned is known as processing delay. The task length (TL) to virtual machine capacity ratio is determined by dividing the task length (TL) in millions of instructions (MI) by the VM processing in millions of instructions per second (MIPS), as shown in Eq 3.1. (MIPS). The processing formulation is as follows:

$$ED = \frac{TLen}{VMMIPS} \qquad \textit{Eq 3. 1}$$

Where ED is Execution Delay, TLen is task length, VMMIPS is virtual machine capacity

Transmission Delay (T_d): The amount of time it takes for data to be transferred from one node to another node at a given bandwidth rate is referred to as transmission delay. As a result, task size (IN/Bw+OP/Bw) and bandwidth rate (Bw) are required to calculate transmission delay. Using the equations below, transmission delay ($\mathbf{T_d}$) is defined as the ratio of packet size to bandwidth rate at each node.

I. Transmission Delay from IoT devices to nearest(low-level) fog nodes

Transmission delay over a given link between an IoT device and the nearest fog node can be calculated as follows:

$$\mathbf{T_{dn}} = \frac{IN}{BwIn} + \frac{OP}{BwOn} \qquad \textit{Eq 3. 2}$$

Where IN is the task's input file size from IoT, OP is the task's output file size from nearest fog nodes, BwIn is the bandwidth rate from IoT device to nearest fog nodes, BwOn is the bandwidth rate from nearest(low-level) fog nodes to IoT device, and T_{dn} is the transmission delay.

II. Transmission Delay from nearest to aggregate(upper-level) fog nodes

Transmission delay over a given link between the nearest fog and aggregate fog nodes can be calculated as follows:

$$\mathbf{Tda} = \frac{IN}{BwIa} + \frac{OP}{BwOa} \qquad \textit{Eq 3. 3}$$

Where IN is the input file size of the task, OP is the output file size of the task, BwIa is bandwidth rate from the nearest fog nodes to the aggregate fog nodes, BwOa is the bandwidth rate from the aggregate fog nodes to the nearest fog nodes.

III. Transmission Delay from aggregate fog nodes to the cloud nodes

Transmission delay over a given link between the aggregate fog and cloud nodes can be calculated as follows:

$$\mathbf{T}dac = \frac{IN}{BwIc} + \frac{OP}{BwOc} \qquad \text{Eq 3. 4}$$

Where IN is the task's input file size, OP is the task's output file size, BwIc is the bandwidth rate from the aggregate fog nodes to the cloud nodes, and BwOc is the bandwidth rate from the cloud nodes to the aggregate fog nodes.

IV. Data transfer Delay for task processed at upper-level(aggregate) fog nodes

When data is sent from IoT and executed in aggregate fog nodes, the data transfer delay can be calculated using the following equation:

$$Data\ Delay\ (Dda) = \frac{IN}{BwIn} + \frac{IN}{BwIa} + \frac{OP}{BwOn} + \frac{OP}{BwOa} \qquad \text{Eq 3. 5}$$

Where BwIn is bandwidth rate from IoT device to the nearest fog nodes BwOn is bandwidth rate from the nearest fog nodes to IoT devices BwOa is bandwidth rate from aggregate fog node to the nearest fog and BwOn is bandwidth rate from the nearest fog nodes to aggregate fog nodes. IN and OP are input and output file sizes of the task.

V. Data transfer delay for task processed at cloud nodes

When data is sent from IoT and get processed in cloud nodes, it is computed as follows:

$$Data\ Transfer\ Delay\ (Dtdc = \frac{IN}{BwIn} + \frac{IN}{BwIc} + \frac{OP}{BwOn} + \frac{OP}{BwOc} \qquad \text{Eq 3. 6}$$

Where BwIn is bandwidth rate from IoT device to the nearest fog nodes, BwOn is bandwidth rate from the nearest fog nodes to IoT devices, BwIc is bandwidth rate from nearest node to the cloud nodes and BwOc is bandwidth rate from the cloud nodes to nearest fog nodes, IN, and OP are input and output file size of the task.

$$\textbf{Total Transmission Delay } (T_{Td}) = Tdn + Tda + Tdac + Dda + Dtdc \quad \textit{Eq 3. 7}$$

Service Latency

The service latency is the time it takes between when a task is sent to a node and when it is returned to the end device after processing. Its value varies depending on the processing layer used to carry out a task. Because tasks can be performed by nearest(low-level) fog nodes, aggregate(upper-level) fog nodes, or cloud servers, the service latency can be calculated based on the layer where the task is performed. To compute the service latency required to complete a given task, queue delay, processing delay, propagation delay, and transmission delay must all be identified. We can calculate the service delay mathematically as follows:

I. **Service latency when processed at low-level(nearest) fog nodes**

$$S_{ln}=W_{qn}+ED_n+ Pd_{pron} +Td_n \quad \textit{Eq 3. 8}$$

where S_{ln} is the service latency when tasks are executed at the low-level (nearest)fog node, W_{qn} is the task's waiting time to get service into the queue at the nearest-fog node, ED_n is the task's execution time at nearest fog node, Pd_{pron} is propagation delay between IoT to nearest fog nodes and Td_n is the transmission delay from IoT device to low-level(nearest) fog node and vice versa.

II. **Service latency when processed at upper-level(aggregate) fog nodes**

$$S_{la}=W_{qa}+E_{Da}+Pd_{proa}+Td_a \quad \text{Eq 3. 9}$$

where S_{la} is the service latency when tasks are executed at the upper-level (aggregate)fog node, W_{qa} is the task's waiting time to get service into the queue at the aggregate-fog node, ED_a is the task's execution time at aggregate fog node, P_{prona} is propagation delay between IoT to aggregate fog nodes, and Td_a is the transmission delay from low-level(nearest) to upper-level(aggregate) fog node and vice versa.

III. **Service latency when processed at cloud nodes**

$$S_{lc}=W_{qc}+ED_c+ Pd_{proc} +Td_c \quad \text{Eq 3. 10}$$

where S_{lc} is the service latency when tasks are executed at the cloud node, W_{qc} is the task's waiting time to get service into the queue at the cloud node, ED_c is the task's execution time at

cloud node, $\mathbf{Pd_{pronc}}$ is propagation delay between IoT to cloud fog nodes, and $\mathbf{Td_c}$ is the transmission delay from low-level (nearest) fog node to cloud node and vice versa.

$$\mathbf{T_{SL}} = \boldsymbol{Wq} + \mathbf{S_{ln}} + \mathbf{S_{la}} + \mathbf{S_{lc}} \quad \textit{Eq 3. 11}$$

Where $\mathbf{T_{SL}}$ is the total service latency in all layers. $\mathbf{S_{ln}}$ is service latency in nearest(low-level) fog node, $\mathbf{S_{la}}$ is service latency in aggregate(upper-level) fog node, $\mathbf{S_{lc}}$ is service latency in cloud nodes, $\boldsymbol{Wq}$ is the total queue average of waiting time

Cost

The cost is the price paid to perform the task on the selected nodes, which is related to both the node capacity and the price associated with its usage. The cost of transferring a task from one node to another can be calculated using the time it takes the processor to complete the task, the required bandwidth, and the cost of memory usage. When a task is completed, costs are incurred that must be minimized in order to meet the needs of the user. calculating costs for processing, VM utilization, and transmission may be done using the following equations.

Costs of Processing (Cp) The price of the VM and the time it takes to process the task on it determine how much it will cost to process the task. The type of VM affects how much it costs. Here, the storage space and capacity are used to categorize the type of virtual machine. There are three different VM kinds in this work.

$$\boldsymbol{Cp} = \frac{\boldsymbol{TLen}}{\boldsymbol{VMMIPS}} \boldsymbol{x}\, \boldsymbol{\rho c} \quad \textit{Eq 3. 12}$$

Where $\boldsymbol{\rho c}$ is the price of executing a unit of task on specified virtual machines and **Cp** is costs of processing? **TLen** is task length in millions of instructions and $\boldsymbol{VMMIPS}$ virtual machine capacity million instruction per second.

Transmission Cost ($\mathbf{T_{Ct}}$) (Eq 3.14) illustrates how the size of the task's input and output files as well as the cost of bandwidth are used to calculate the input and output transfer cost for each task. A task on a given bandwidth can be calculated as shown below.

$$\mathbf{T_{Ct}} = \frac{\boldsymbol{Data\ size}}{\boldsymbol{Bw}}\ \boldsymbol{x\rho} \quad \textit{Eq 3. 13}$$

Where ρ it's cost of transferring a unit of task on a given bandwidth. **Data size** is input and output file of a task in KB or MB, Bw (**bandwidth rate)** is the number of task sizes in KB or MB that could be transferred over a given link.

$$\textbf{Total cost (TC)} = \boldsymbol{Cp} + \boldsymbol{Tct} \quad \textit{Eq 3. 14}$$

Task Deadline

The research's main objective is to minimize latency and cost on assigned nodes while maintaining task deadlines. According to(Byers, 2017), tasks with latency requirements of less than a few tens of seconds require a special technique to get a response in time. The task deadline is used as a parameter to evaluate and decide where a task should be processed. The expected total service latency at the time of arrival should be less than the deadline time. The task will be assigned at the node if the deadline is meeting. Otherwise, the best neighboring nodes are chosen.

Table 3 2 Definition Table

T_d	Transmission Delay	TL_{dn}	Transmission delay from IoT devices to nearest fog nodes
ED	Execution Delay	TLda	Transmission delay from nearest to aggregate fog nodes
PD	Propagation Delay	T_{Len}	Task Length
T_{Ct}	Transmission cost	Su	system utilization
Cp	Costs of Processing	Wq	average queuing latency
TC	Total cost	S_{ln}	Service latency when processed at low-level(nearest) fog nodes
VMC	Virtual machine usage cost	S_{la}	Service latency when processed at upper-level(aggregate) fog nodes
Dda	Data transfer delay for task processed at aggregate fog nodes	S_{lc}	Service latency when processed at cloud nodes
$Dtdc$	Data transfer delay for task processed at cloud nodes	Bw	bandwidth
T_{SL}	Total Service Latency	$Tdac$	Transmission delay from aggregate fog nodes to the cloud nodes

3.4. Grey Wolf optimization (GWO)Algorithm

The gray wolf optimization algorithm simulates the leader hierarchy and predation behavior of wolves, then uses the gray wolf's ability to search, surround, and hunt during predation for optimization purposes. Suppose the number of wolves is N, the search area is d, and the position of the first wolf can be expressed as follows. Xi = (Xi1, Xi2, Xi3.), Xid). For the purpose of mathematical modeling of wolf social hierarchy, the alpha (α) wolf is considered as the most appropriate solution. Therefore, the second and third best solutions were called β and δ wolves, respectively. The remaining candidate solutions were considered as omega (ω) wolves. In this

algorithm, the position of the prey corresponds to the position of the alpha wolf. In this study grey wolf represents virtual machine and prey represent task.

the following matrix describes the location of grey wolves in vector form.

$$\begin{bmatrix} GW0,0\ ,GW\ 0,1\ ,GW0,2\ ,GW\ 0,3\ ,\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots,GW0,j \\ GW1,0\ ,GW\ 1,1\ ,GW\ 1,2\ \ GW1,3\ ,\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots,GW1,j \\ GW2,0\ ,GW\ 2,1\ ,GW\ 2,2\ ,GW\ 2,3,\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots,GW2,j \\ \vdots \\ GWi,0\ \ ,GW\ i,1\ ,GWi,2\ \ ,\ \ GW\ i,2,\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots GW3,j \end{bmatrix} \quad \textit{Eq 3. 15}$$

Each row in ith dimension is assumed to be a grey wolf predator for food. Each value of GWi, j dimension represents the virtual machine to be allocated for the task that is represented as 0 to j index where each value represents the position of each task length in the task array. When each row is evaluated by fitness function as shown in (Eq 3. 16), the one with the most minimum fitness value will be the alpha i.e., row 0 in (Eq 3. 16). The second, third, and fourth fitness values in order will be assumed as beta, delta, and omega which is rows 1, 2, and 3 in (Eq 3. 16) while the rest are considered as omega. The whole algorithm contains three main stages: encirclement, hunting, and attack. Its specific process is as follows.

Encircling prey Stage

The main purpose is to identify the target prey and then encircle them. The following equations are suggested to mathematically model encircling behavior:

$$\vec{D} = |\vec{C}.\vec{X}p(t) - \vec{X}(\mathrm{t})| \quad \textit{Eq 3. 16}$$

Where $\vec{X}p$ is the position vector of the prey, and $\vec{X}(t)$ indicates the position vector of a grey wolf. $\vec{D}$ denotes the distances between the position vectors both of the prey ($\vec{X}p$) and a wolf ($\vec{X}(t)$).

The position transformation formula of the gray wolf is as follows

$$\vec{X}(t+1) = \vec{X}p(t) - \vec{A}\ .\vec{D} \quad \textit{Eq 3. 17}$$

$$\vec{\mathrm{A}} = 2\,\vec{a}.\vec{r}1 - \vec{a} \quad \textit{Eq 3. 18}$$

$$\vec{a} = 2\left(1 - \frac{t}{T\,max}\right) \quad \textit{Eq 3. 19}$$

$$\vec{C} = 2\vec{r}2 \quad \textit{Eq 3. 20}$$

Where t indicates the current iteration, the set $\vec{A}$ is convergence factor and $\vec{C}$ is coefficient vectors (control coefficient), the components of $\vec{a}$ decrease linearly from 2 to 0 during the iteration, and r1, r2 are random vectors in [0, 1]. amax=2, amin=0

When the gray wolf captures a prey, the leader wolf α first leads the other wolves to surround the prey. Then wolf α leads wolves β and δ to capture the prey. Among the grey wolves, α, β and δ are the ones closest to the prey, so the position of the prey can be calculated from their positions. We assume the alpha, beta, and delta have superior information of the probable location of prey in order to mathematically simulate the hunting behavior.

Hunting Prey Stage

after the encirclement, wolves α will lead wolves β and wolves δ to hunt, while wolves ω will be guided by wolves α, β and δ to update position. Where $\vec{X}(t+1)$ is the optimal solution for i + 1 iterations, and the mathematical model of specific location information transformation is as follows.

$$\vec{D}\alpha = |\vec{C}1.\vec{X}\alpha - \vec{X}(\mathrm{t})| \quad \textit{Eq 3. 21}$$

$$\vec{D}\beta = |\vec{C}2.\vec{X}\beta - \vec{X}(t)| \quad \textit{Eq 3. 22}$$

$$\vec{D}\delta = |\vec{C}3.\vec{X}\delta - \vec{X}(\mathrm{t})| \quad \textit{Eq 3. 23}$$

$$X1 = X\alpha - A1\,x\,\vec{D}\alpha \quad \textit{Eq 3. 24}$$

$$X2 = X\beta - A2\,x\vec{D}\beta \quad \textit{Eq 3. 25}$$

$$X3 = X\delta - A3\,x\,\vec{D}\delta \quad \textit{Eq 3. 26}$$

$$\vec{X}(t+1) = \frac{X1+X2+X3}{3} \quad \textit{Eq 3. 27}$$

where C1, C2, C3 are coefficients that can be calculated according to (Eq 3.21) to obtain specific values. The distance between *X(t)* and *α, β,* ω wolves is calculated by equation (Eq 3. 22)– (Eq 3. 27), and then, the position of the wolves move to the prey is calculated by equation (Eq 3. 28). As was already indicated, after the prey stops moving, the grey wolves attack it to end the hunt.

Attack stage: The target is captured, and the optimal solution is obtained. As the convergence factor a decreases linearly from 2 to 0, the value of A varies within [−2, 2]in (Eq 3.18) and (Eq 3.19). When $|A| \geq 1$, it means that it is still in the global search stage; while $|A| < 1$, the wolf pack will launch an attack to capture the targets which are already locked

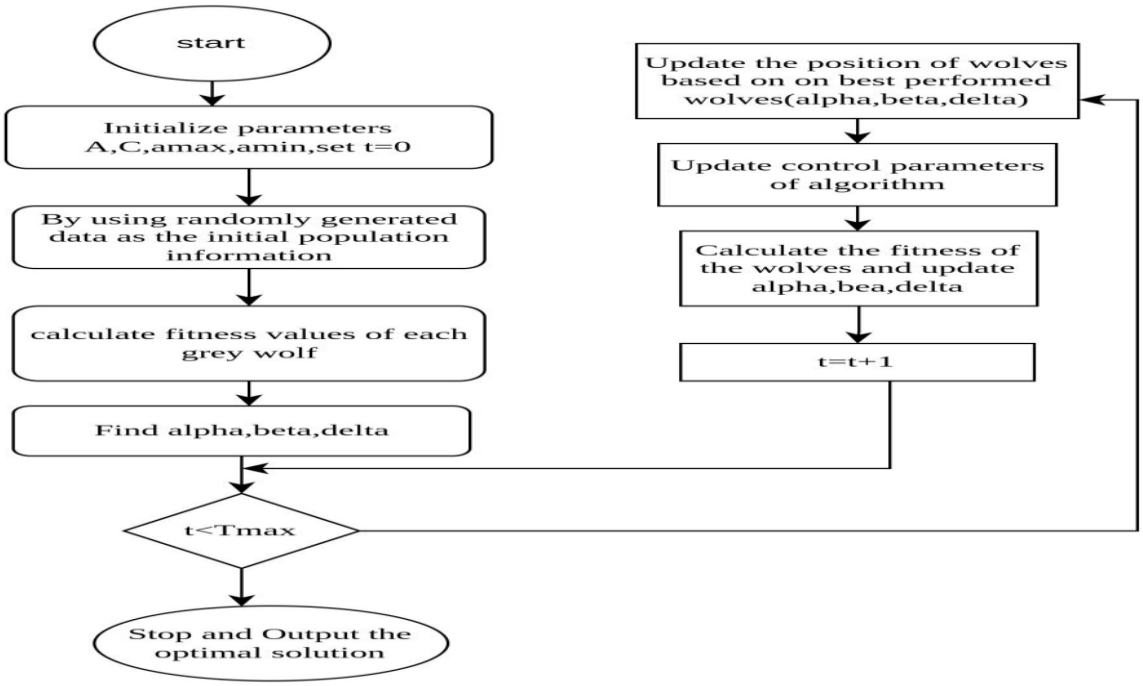

Figure 3. 4 GWO algorithm optimization process

3.5. **Improved hybrid Grey Wolf optimization (IHGWO)Algorithm for scheduling**

The existing Grey Wolf optimization algorithm has a few disadvantages, such as a slow convergence rate, poor precision, and others. The strength of the exploitation ability of the particle swarm optimization (PSO) and the exploration ability of the grey wolf optimizer (GWO) were combined to improve this limitation. This novel technique uses the Tent chaotic sequence to start the positions of the individuals, which can increase the diversity of the wolf pack. The nonlinear control parameter is also used to improve convergence and balance the local and global search capabilities of the algorithm. At the same time, the concept of PSO is presented, which updates each grey wolf's position information by using the best values of the individual and the best values of the wolf pack. This method preserves the best position information of the individual and avoids the algorithm falling into a local optimum.

Chaos Initialization

The GWO algorithm typically uses randomly generated data as the starting population information for solving function optimization problems, which will cause the population's variety to be lost and result in poor optimization performance. In order to initialize the population, we have used a Tent chaotic map.

The Tent Chaos Map's mathematical model is as follows

$$X(t+1) = \begin{cases} \frac{x(t)}{u} & 0 \leq x(t) < u \\ \frac{1-x(t)}{1-u} & u \leq x(t) \leq 1 \end{cases} \quad \textit{Eq 3. 28}$$

The Tent map has its most frequent shape when u = 1/2. The resulting sequence at this point has a uniform distribution and a roughly uniform distribution density for the various parameters.

The Tent chaotic map formula used in this study is as follows:

$$Xt+1 = \begin{cases} 2x_t & 0 \leq x_t \leq LB \\ 2(Xt-1) & LB \leq x_t \leq UB \end{cases} \quad \textit{Eq 3. 29}$$

Where LB is lower bound and UB is upper bound.

Nonlinear Control Parameter Strategy

The prey positioning and the individual grey wolf's hunting behavior make up the majority of the GWO algorithm. The global exploration and local exploitation abilities of the GWO algorithm are balanced, according to (Eq. 3. 17), in a significant way by the parameter A. Because of the GWO algorithm's capacity to perform global exploration, when |A| > 1, the group will expand its search area to locate a better candidate solution. The GWO algorithm has the ability to use local resources when |A|< 1, which causes the group to focus its search range and conduct detailed local searches. At the same time, we can observe from (Eq 3.18) that during iteration, the value of A changes continually as the control parameter a is changed. The control parameter a decrease linearly with the increase in the number of iterations, as can be observed from (Eq. 3. 19). However, the GWO algorithm’s optimization procedure is exceedingly challenging. The algorithm’s true optimization search process cannot be captured by a parameter’s linear change. Therefore, as indicated in the following equation, suggested a new set of nonlinear control parameters.so we have used the following (Eq 3.31) for this study for the purpose of focusing in local research details because our aim is scheduling in the nearest

optimal solution of node if the optimal node is not available it will be search for global which by starting from local to global.

$$\boldsymbol{a} = \boldsymbol{aini} - (\boldsymbol{aini} - \boldsymbol{afin})\ \boldsymbol{x}\ \left(\frac{\boldsymbol{t}}{\boldsymbol{Tmax}}\right)^{\mathbf{2}} \qquad \textit{Eq 3. 30}$$

where the set ***a*ini** and ***a*fin**, respectively, represent the initial value and final value of the control parameter *a*. The set *t* is current iteration and the set *T*max is the maximum number of iterations.

Particle swarm optimization (PSO) Thought

The GWO algorithm only considers the location information of individual wolves and the best, second-best, and third-best location information of the wolf pack while updating positions, realizing the information exchange between individuals and the wolf pack. However, it neglects the wolf's informational exchange and its own experience. As a result, the PSO algorithm concept is presented in order to enhance the process of updating positions. The best position information for the particle and the best position information for the group are used in the PSO method to update the particle's current position. The new location update equations are as follows:

$$\mathbf{xi}(\boldsymbol{t}+\mathbf{1}) = \boldsymbol{c1r1}\big(\boldsymbol{w1X1}(\boldsymbol{t}) + \boldsymbol{w2X2}(\boldsymbol{t}) + \boldsymbol{w3X3}(\boldsymbol{t})\big) + \boldsymbol{c2r2}\big(\boldsymbol{Xibest} - \boldsymbol{Xi}(\boldsymbol{t})\big) \qquad \textit{Eq 3. 31}$$

The first part of (Eq 3. 31), which expands the search interval to improve the algorithm's ability to search globally, is expressed as the mean value of the best prey position using α, β, δ wolves. The second part, which preserves the individual's optimal position, is expressed as the effect of the individual's historical best position on the algorithm. The set c2 is a cognitive learning factor, as opposed to the set c1 which is a social learning component. They stand for the influence of the group optimal value and the individual optimal value, respectively. The huge value of c1 can enhance global search capabilities, and the large value of c2 can enhance local search capabilities. But if the c1 is too large, it will result in too many particles remain in the vicinity of the local. If the c2 is too large, it will cause the particles to reach the local optimum in advance and converge to this value.in this studies we have used c1 = c2 = 2.05. The set r1 and r2 are the random variable in the range of [0, 1]. The set Xibest indicates that the grey wolf has experienced the best position. The set w1, w2, w3 are inertia weight coefficients. By adjusting weight ratio of the α, β, δ wolves, the global and local search ability of the algorithm can be balanced dynamically. The specific equation is as follows:

$$\boldsymbol{w1} = \frac{|\boldsymbol{X1}|}{|\boldsymbol{X1}+\boldsymbol{X2}+\boldsymbol{X3}|} \qquad \textit{Eq 3. 32}$$

$$\boldsymbol{w2} = \frac{|\boldsymbol{X2}|}{|\boldsymbol{X1}+\boldsymbol{X2}+\boldsymbol{X3}|} \qquad \textit{Eq 3. 33}$$

$$\boldsymbol{w2} = \frac{|\boldsymbol{X2}|}{|\boldsymbol{X1}+\boldsymbol{X2}+\boldsymbol{X3}|} \qquad \textit{Eq 3. 34}$$

3.6. **Task to Virtual mapping using proposed algorithm**

The task scheduling problem is one of efficiently mapping tasks to available resources while satisfying both user and provider requirements. Each grey wolf in the proposed algorithm represents a possible solution for allocating tasks to the available resources, which in our case is a virtual machine. The proposed algorithm is significant in that it achieves a deadline with minimal latency and minimal costs of tasks to be executed on a virtual machine, allowing both user and provider requirements to be met. To accomplish this, the objective function must be defined in such a way that the proposed algorithm can easily apply to the problem. The objective function is the summation of each task Ti executed on each virtual machine VMij with cost and latency associated with it as the following equations.

$$minf(x) = \sum_{j=1}^{n} (\, 1 - \alpha \;\; \text{x TotalCosts} \; + \;\; \alpha \;\text{x TotalLatency}) \qquad \textit{Eq 3. 35}$$

Total Costs is the cost of execution, usage of VM cost, and transmission costs at node VMij. The lower objective function is the better wolves close to prey. Latency must meet the deadline of a task. Where α ($\alpha \in [0, 1]$) is the balance coefficient between latency and total cost. $\alpha = 0.5$ means that time and cost have same priority in optimizing. When > 0.5, our approach prioritizes minimizing latency over total cost, indicating that a user is prepared to pay more for better performance. Inversely, when $\alpha < 0.5$, the cost is more prioritized than time, i.e., the user has a tight budget. for this work we have taken α=0.5.

The following are the specific implementation steps for the improved hybrid grey optimization wolf algorithm:

1. Begin
2. Set the size of the task, delay_insensitive, and initialize the A, C, a value;

3. Generate population individuals by using Tent mapping {Xi, i = 1, 2, 3. . . N}, then calculate the individual fitness value {fi, i = 1, 2, 3. . . N};
4. Sort the order of fitness values by size, and take the first three fitness values corresponding to the individual as α, β, δ. The corresponding position information is: Xα, Xβ, and Xδ;
5. Use formula (Eq 3. 31) to calculate the nonlinear control parameters a, and then update the value of A and C according to (Eq 3.19) and (Eq 3. 21).
6. Use formula (Eq 3. 32) to update the location of individuals, then recalculate the fitness values and update values of α, β, δ;
7. Judge whether t reaches Tmax value, if reached, the fitness value of α is output, that is, the best solution.

Else go to Step 4.

Improved hybrid Grey Wolf Optimizer (IHGWO)Algorithm Pseudo-Code

Input: number_of_task, delay_insensitive, delay_sensitive, delay_less_sensitive, number of iterations Tmax, Control parameter a, learning factor c1, c2

Output: Optimal solution

1. for i= 1 → N do
2. for j= 1 → D do
3. use equation (Eq 3. 30) to generate Tent chaotic sequence
4. End For
5. End For
6. for t<=Tmax do
7. Use equation (Eq 3. 31) to update parameter a value
8. for i= 1 → N do
9. for j= 1 → D do
10. D=|CXp(t)-X(t)|, then according to equation (Eq 3. 22) (Eq 3. 23) (Eq 3. 24) to calculate D (α, β, δ)
11. According to equation (Eq 3. 33) (Eq 3. 34) (Eq 3. 35) to calculate the coefficient **w**
12. According to equation (Eq 3. 25) (Eq 3. 26) (Eq 3. 27) (Eq 3. 32) to update the position of the grey wolf

13. End For
14. End For
15. calculate fitness values and update α, β, δ
16. end // the algorithm ends

3.6.1. **Proposed Algorithm Flowchart**

the proposed algorithm's flowchart, in which the first tasks are classified based on their deadlines using (Byers, 2017)threshold values and task size after the task has been classified, it must be kept in a queue until the efficient node is chosen by the node selection algorithm. Tasks are assigned to latency-sensitive, latency-tolerant, and latency-insensitive tasks based on their arrivals at the queue. The first task that arrives must be completed first because it has a very short deadline, which can be measured in microseconds. If a task's service latency is not met by the deadline in the low-level fog node, the task must be assigned to the upper-level fog node if service latency at upper-level fog nodes becomes valid for a task. Furthermore, tasks at latency-tolerant and medium task size must be assigned to upper-level fog nodes. However, if the service latency of the task on the available resource in upper-level fog nodes does not meet the task deadline as well as capacity, the task should be assigned to the cloud node if the service latency at the cloud node becomes valid, otherwise it should be assigned to an efficient node that can execute tasks near deadlines.

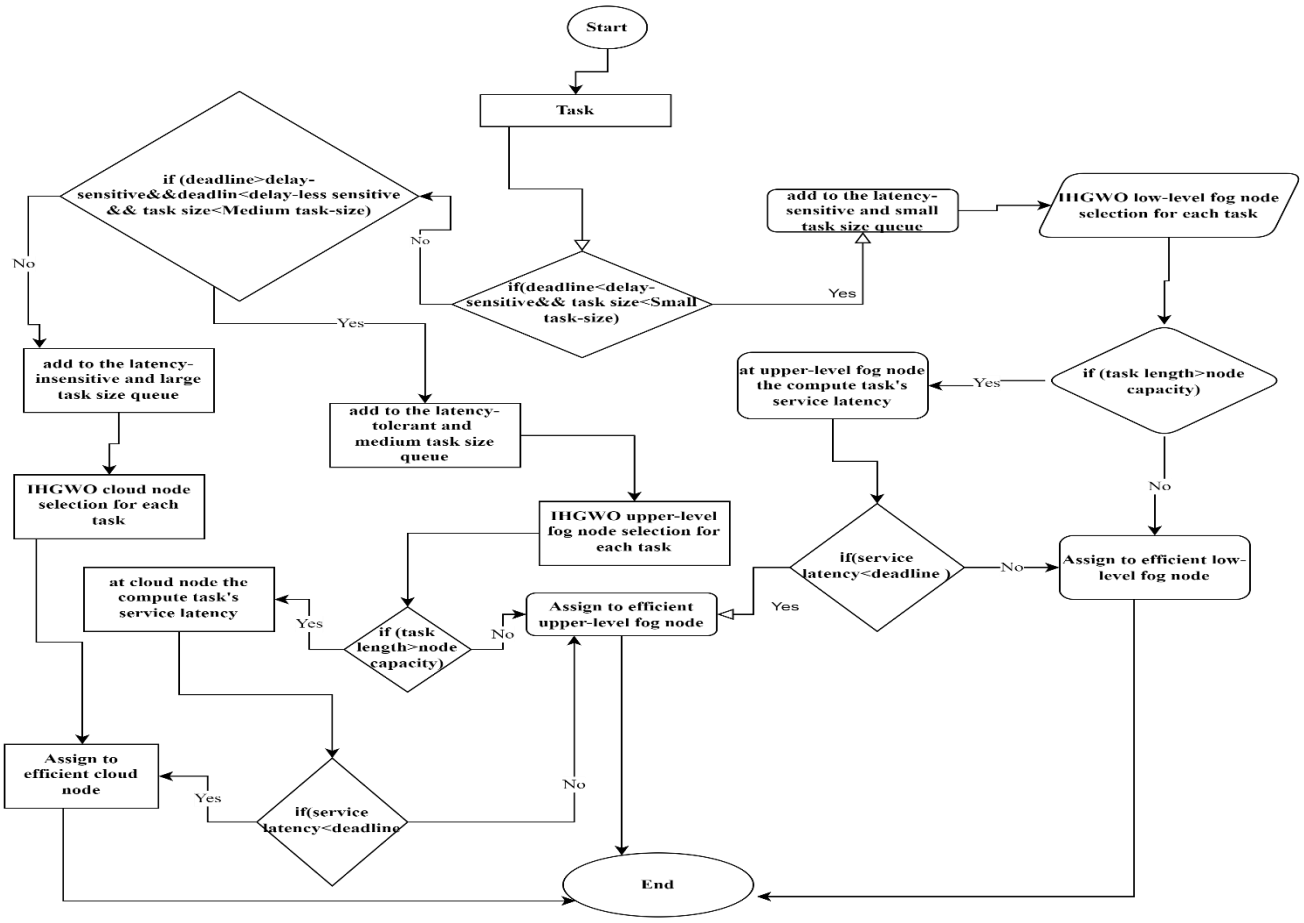

Figure 3. 5 flow charts of the proposed algorithm

Pseudo Code of Task Allocation Algorithm

Attributes of input

Task list with task size and deadline attributes

Output

Specific type tasks into latency-sensitive and small data-size tasks, latency-tolerant and medium data-size tasks, and latency-insensitive and large data-size tasks.

The task arrives with the attribute's deadline and task-size.

Add deadline and task size threshold value one (T_{d1} and T_{s1})

Add deadline and task size threshold value two (T_{d2} and T_{s2})

For each task in the queue task

If (deadline of task<T_{d1}&&task-size<T_{sl})

Add task in latency-sensitive queue

Else If (deadline of task>T_{d1} && deadline of task<T_{d2}||task-size<T_{s2})

Add task in latency-tolerant queue

Else

Add task in latency-insensitive queue

Endif

End For

Algorithm for Node Controllers

Node controller is a component that keeps the information about the current status of every node such as waiting time.

Input: Task arrived with average arrival rate λi

Output: Waiting time at each Node i

Incoming Task i

Get previous waiting time W_{tp}, which before task arrived

Calculate execution time of a task

Compute waiting time at the Node:

$W_t = W_{tp} + ED_{avg}$

Update node waiting time status W_{tp}

End

Meta-heuristic search algorithm for node selection for latency-sensitive tasks

The meta-heuristic search algorithm searches for appropriate nodes for a latency-sensitive task that meets the deadline and task size.

Input

Task-size, task-deadline in latency-sensitive and small task-size queue

A virtual machine is available at the low-level fog node.

Output

Allocated node

1. For each task in latency-sensitive and small task-size queue
2. Assigned = false
3. For each node i=1, to n in the low-level fog node do
4. Calculate service latency of task in node i

5. $\mathbf{S_{ln}=W_{qn}+ED_n+ Pd_{pron} +Td_n}$
6. If (S_{ln}<deadline)
7. Assigned task at node i
8. Calculate S_{ln} with different available cost of node i processor
9. Assign min cost to node i processor if<S_{ln} deadline to save cost
10. Assigned = true
11. Endif
12. End For
13. If (! Assigned)
14. For each node m=1, to n in upper-level fog node do
15. Calculate service latency of task in node m'
16. $\mathbf{S_{la}=W_{qa}+ED_a+Pd_{proa}+Td_a}$
17. Endif
18. If (S_{la}<deadline)
19. Assigned task at node m
20. Calculate $S_{la}$with different available cost of node i processor
21. Assign min cost to node i processor if<S_{la} deadline to save cost
22. Assigned = true
23. Endif
24. End For
25. End For

Node Choice Meta-Heuristic Search Algorithm for Latency Tolerant(less-sensitive) Task

The Meta-Heuristic Search Algorithm also finds a node for latency-tolerant(less-sensitive) tasks. Rather than assigning the first node found that meets the deadline, as in latency-sensitive tasks, the latency-tolerant(less-sensitive) task searches the aggregate (upper-level) fog node for the best node in terms of latency and cost and assigns the task to the most efficient node found. The pseudo-code that follows describes node selection at the aggregate (upper-level) node.

Input

Deadline of a task, size of a task in latency-tolerant and medium task size queue

Virtual machine available at upper-level fog node

Output

Allocated node

1. For each task in latency tolerant and small task-size queue
2. Assigned = false
3. For each node i=1, to n in the upper-level fog node do
4. Calculate service latency of task in node i
5. $\mathbf{S_{la}=W_{qa}+ED_a+Pd_{proa}+Td_a}$
6. If (S_{la}<deadline)
7. Assigned task at node i
8. Calculate $S_{la}$with different available cost of node i processor
9. Assign min cost to node i processor if<S_{la} deadline to save cost
10. Assigned=true
11. End if
12. End for
13. If (! Assigned)
14. For each node m=1, to n in the cloud node do
15. Calculate service latency of task in node m
16. $\mathbf{S_{lc}=W_{qc}+ED_c+ Pd_{proc} +Td_c}$
17. If ($\mathbf{S_{lc}}$<deadline)
18. Assign task at node m
26. Calculate S_{lc} with different available cost of node i processor
27. Assign min cost to node i processor if<S_{lc} deadline to save cost
19. Assigned=true
20. End if
21. End for

Meta-heuristic search algorithm for node selection for latency-insensitive tasks

The meta-heuristic search algorithm searches for appropriate nodes for a latency-insensitive task that meets the deadline and task size.

Input

Task-size, task-deadline in latency -insensitive and large task-size queue

A virtual machine is available at the cloud node.

Output

Allocated node

1. For each task in latency-insensitive and large task-size queue
2. Assigned = false
3. For each node i=1, to n in the cloud node do
4. Calculate service latency of task in node i
5. $\mathbf{S_{lc}=W_{qc}+ED_c+ Pd_{proc} +Td_c}$
6. If (S_{ln}<deadline)
7. Assigned task at node i
8. Calculate S_{lc} with different available cost of node i processor
9. Assign min cost to node i processor if<S_{lc} deadline to save cost
10. Assigned = true
11. Endif
12. End For

CHAPTER FOUR

4. Experiment Results and Discussions

This also discusses the simulation setup and task scheduling process simulation, as well as the task parameter, resource parameter, and outcome evaluation metrics including cost, latency, and completion time reduction. With an improved hybrid grey wolf optimization algorithm, we were compared to other algorithms including First Come First Serve (FCFS), Short Job First (SJF), Genetic Algorithm, Grey Wolf Optimization, and Particle Swarm Optimization (PSO). The comparison was conducted using the Cloudsim simulator to verify its effectiveness. The simulation was conducted using Java and the NetBeans development environment.

4.1. Research Simulation Tools

Cloudsim enables researchers to focus on specific system design issues rather than worrying about low-level details associated with cloud-based infrastructures and services. It is a java-based toolkit that includes actors or entities that are nothing more than java classes that communicate with each other during the simulation process(Humane & Varshapriya, 2015).
Cloudsim have the following common entity or actors which performed on action

1. **Virtual machine (VM)** is logical machine on which application or task will be run.
2. **Virtual machine Generator (VMG)** these generators the VMs on Cloudsim Simulator and submit all VMs to the broker
3. **Data center (DC)** this provisions the resources, and may have one or more than one hosts
4. **Host** is a physical machine on which logical machines are placed
5. **Cloud Information Services (CIS)** an entity which manages all registering of resources
6. **Broker** is an agent, who will submit the VMs in specific host and then bind cloud-lets, applications, processes on specific VMs and after execution of all the cloud-lets, the free VMs will be automatically destroyed.
7. **Data center Broker** an agent who is responsible for communications between data center and VMs as well as Cloud-lets
8. **Cloud-lets** are the task or application that run on VMs

9. **Cloud-lets Generator** will generate the cloud-lets and submit Cloud-let or task list to the broker
10. **BW Provisioner** the provision policy of bandwidth to VMs that are deployed on a Host component
11. **Memory Provisioner** represent the provision for allocation of memory to VMs
12. **VM Allocation Policy** This is an abstract class implemented by a Host component that models the policies (which may be space-shared or time-shared) required for allocating processing power to VMs.

These classes all have corresponding methods that improve the simulation process.

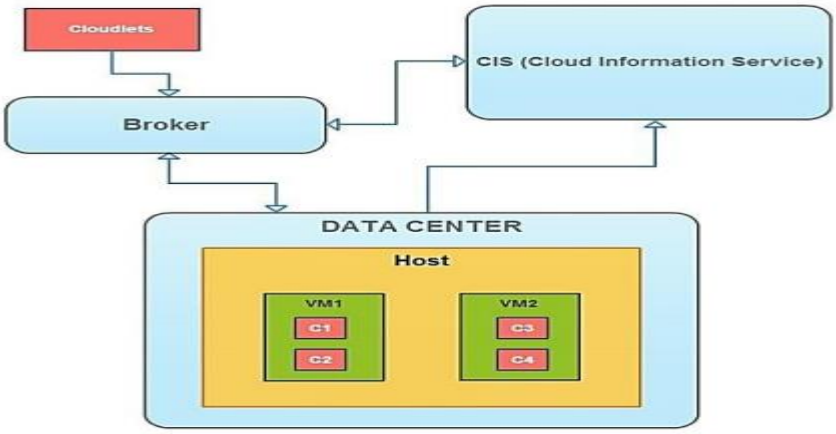

Figure 4. 1Cloudsim Model(Bahwaireth et al., 2016)

4.2. Simulation Model

Figure 4.1 depicts the Cloudsim framework or model. Cloud Information Services (CIS), Data-center, Broker, Virtual Machine (VM1, VM2), Host, and Cloud-lets are its entities (C1, C2, C3, C4). The CIS functions as a registry for all cloud resources. Before they can be used, all cloud resources must be registered with the CIS. There are one or more Hosts in the data center. The following configuration parameters are available for the Hosts: -Processing Element (PEs), RAM, Bandwidth (Bw), and so on. Because this is a virtualized environment, the Virtual Machines will always reside within the Host. The Cloud-lets are executed on the VMs. A broker is a member of the data-center broker class. It is the Broker's responsibility to submit tasks to the Data-center. The broker communicates with the CIS and requests information about registered resources. The data-center characteristics are then returned to the broker by CIS. The broker has Cloud-lets (applications) that have yet to be assigned. Once the broker has received

all of the information from CIS, it submits the Cloud-let List to the Data-center. Cloudsim default allocation policies make the allocation decision (like Cloud-let Scheduler, VmAllocationPolicys VmScheduler etc.). The Hosts (which reside in the data center) schedule the VMs, and the VMs (which reside in the Host) schedule the Cloud-lets.

4.3. Proposed Algorithm Evaluation

A parameter, such as a task attributes or virtual machine attribute, must be clearly described and configured in order to compare the proposed algorithm to other algorithms before evaluati ng the algorithm's performance.

Capacity of virtual machines

To ensure that the heterogeneous nature of cloud and fog computing resembles real-world environments, varying virtual machine capacity has been used. Fifty virtual machines have been created, with 10 virtual machines located on nearest(low-level) fog nodes, 20 virtual machines located on aggregate(upper-level) fog nodes, and 20 virtual machines located on cloud nodes. Based on its location, each virtual machine can have a different capacity, such as nearest fog node, aggregate fog node, and cloud nodes.

Tasks (cloud-lets)

are requests sent by IoT devices to be executed at various levels, such as the low-level fog, aggregate fog layer, and cloud layer. In this study, the proposed algorithm was evaluated using 500 tasks of varying sizes. These tasks have attributes like task size, task size, and execution time that should be assigned to the appropriate nodes to satisfy both users and providers. The table below, for example, allows you to specify a variety of parameters used in this study.

Table 4. 1 Configuration parameters

Parameter	Value
Size of tasks (Million Instructions)	[1-2000]
I/O data of a tasks (KB/MB)	[1-1000] [1-1000]
Deadlines of tasks (ms/sec/min)	[0.1μs -1hr]
Processing rate at nearest(lower-level) fog node (MIPS)	[100-1000]
Processing rate at aggregate(upper-level) fog node (MIPS)	[1000-3000]
Processing rate cloud node (MIPS)	[3000-10000]
Bandwidth in nearest(low-level) fog environment (Mbps)	[20-100]

Bandwidth in aggregate(upper-level) fog environment (Mbps)	[100-200]
Bandwidth in cloud environment (Mbps)	[200-1000]
Processing cost at nearest(low-level) fog per data unit	[0.1-0.2]
Processing cost at aggregate(upper-level) fog per time unit ($)/hr.	[0.2-0.5]
Processing cost at cloud per time unit ($)/hr.	[0.4-0.6]
Transferring cost at cloud environment ($)/ MB	[0.1-0.2]
Transferring cost at nearest(low-level) fog per data unit ($)/MB	[0.001-0.01]
Transferring cost at aggregate(upper-level) fog per data unit ($)/MB	[0.01-0.1]
Transferring cost at cloud environment ($)/ MB	[0.1-0.2]

Network Bandwidth

Because transmission delay has a significant impact on response time, defining bandwidth rate between IoT devices, fog nodes, and between fog nodes is required to specify a proper node allocation. Based on this, the connection between IoT devices and fog nodes is divided into three types. The bandwidth rate between IoT devices that generate small tasks is assumed to be 250 KBps, 25 MBps for IoT devices that generate medium tasks, and 54 MBps for devices that generate large tasks. The transmission rate between fog nodes is 100MBps. Furthermore, the cloud's bandwidth rates can reach 10GBps(Yousef pour et al., 2017).The other experimental parameter setup used is the two-task offloading delay threshold values of 1s and 2minutes, which determine the category of tasks. These threshold values are determined by the application's latency requirement(Byers, 2017).

4.3.1. Comparing the suggested algorithm to existing methods based on latency

Latency is the amount of time it takes for a task to be completed before its deadline. Based on the deadline and task size, the task could be assigned to the low-level(nearest) fog node, upper-level(aggregate) fog node, or cloud node. Detail description of task assignment is available in appendix I, II and III.

Evaluation latency at Low-level fog node

As depicted in figure 4.2, we compared the latency reduction rates of the proposed method to those of five other algorithms: grey wolf optimization, first come first serve, shortest job first,

particle swarm optimization, and genetic algorithm. The suggested approach reduced latency more effectively than other algorithms with various datasets, such as 50, 100, and 150.

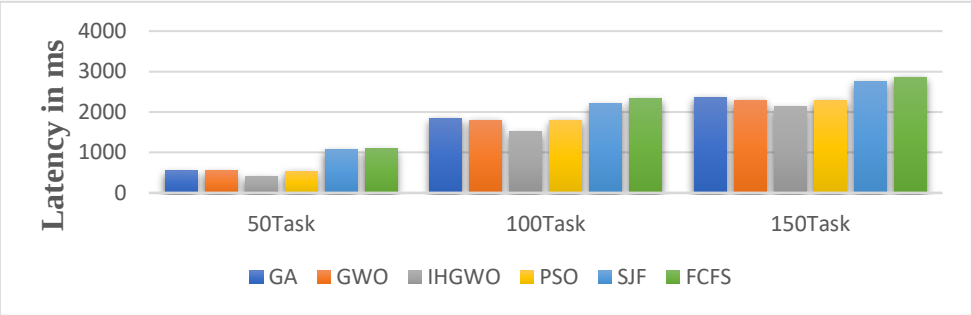

Figure 4. 2 Latency evaluation for low-level fog node task

Evaluation of latency at the upper -level(aggregate) fog node

To achieve the delay requirement, the medium-sized and latency-less sensitive (tolerant) tasks should be executed at the aggregate (upper-level) fog node. We compared the suggested method with heuristics algorithms on different datasets. The suggested method outperformed heuristic algorithms in that it correctly assigned the tasks to the suitable nodes, reduce delay and completed the tasks by the deadline.

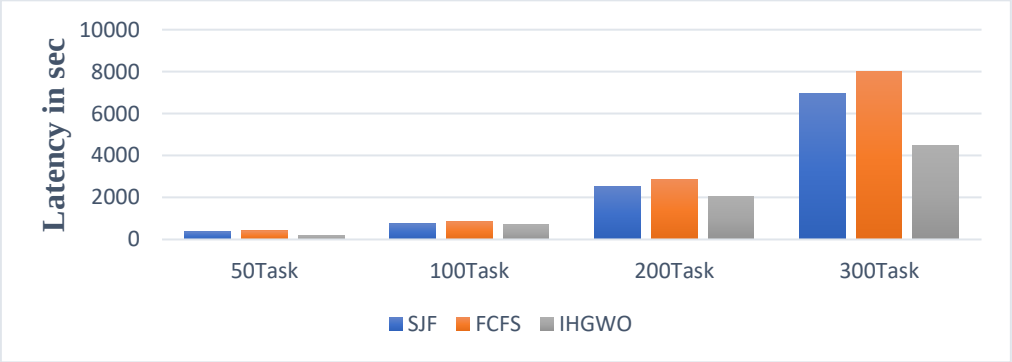

Figure 4. 3 latency evaluation at upper-level(aggregate)fog node

Evaluation Latency at cloud node

The execution of the large task with no latency constraints takes place on a cloud node. We evaluated the latency of the cloud with five alternative strategies, such as first-come first-served short job first, genetic algorithm, particle swarm optimization, and grey wolf optimization, using 150,300 and 500 datasets. The outcomes demonstrate that improved hybrid grey wolf optimization (IHGWO) algorithm outperformed the others approaches.

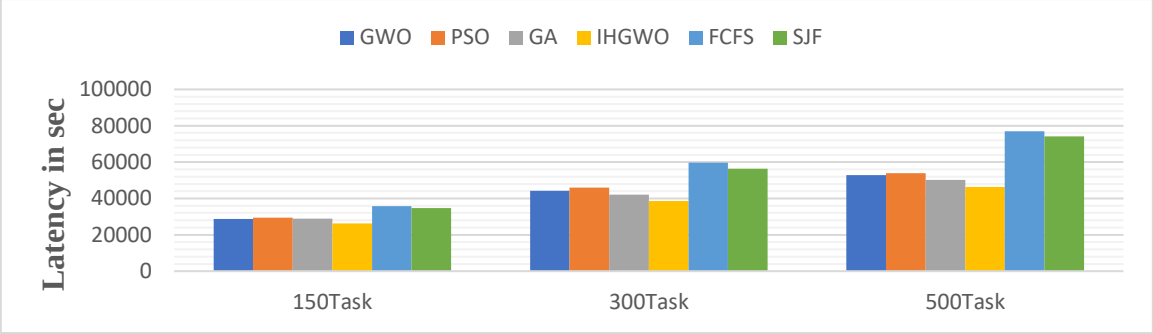

Figure 4. 4 latency at cloud node

Both the shortest job first approach and the first come first served algorithm perform poorly when distributing tasks to the right nodes, as shown in the graph above. In contrast, the suggested algorithm (IHGWO) lowered response time for each task more effectively than other nature-inspired algorithms like PSO, GWO, and GA.

4.3.2. **Evaluation of proposed algorithm based on cost reduction**

In order to meet the demands of its users without exceeding task deadlines, the cost of resource usage and execution cost must be minimized. In this work, we take into account the cost of utilizing virtual machine resources, the cost of execution, and the cost of transferring a task from one layer to another. The proposed algorithm reduced usage costs and execution cost by assigning tasks to low-cost nodes capable of meeting task deadlines. Detail description of task assignment is available in appendix I, II and III.

Cost evaluation at low-level(nearest) fog node

The proposed algorithm has been compared at the low-level fog node with various datasets in terms of costs without conflicting deadlines. The experiment demonstrated that the improved hybrid grey wolf optimization (IHGWO)technique outperformed the others existing approaches.

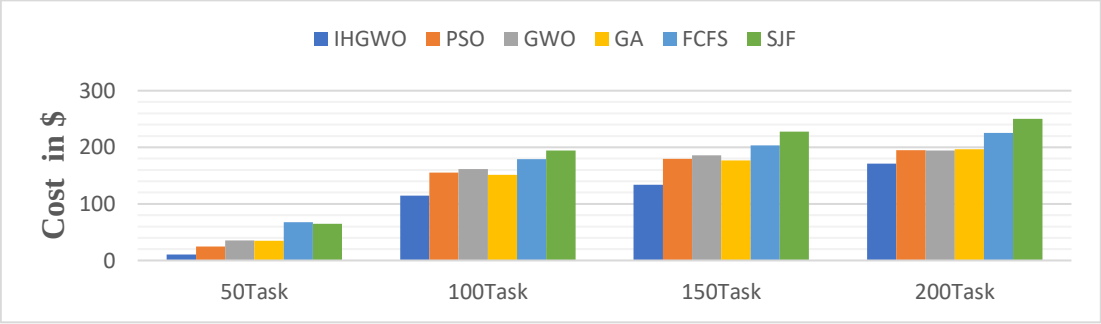

Figure 4. 5 cost evaluation at low-level fog node

Evaluation of cost at Upper-level(aggregate)fog node

In terms of computational power and storage, the low-level(aggregate)fog node exceeds the low-level fog node. When comparing the proposed approach at this layer with the existing approach using different datasets, it was shown that the suggested algorithm performed better than the existing approach as shown figure below. This layer is where medium-sized tasks with latency-less sensitive are assigned and processed.

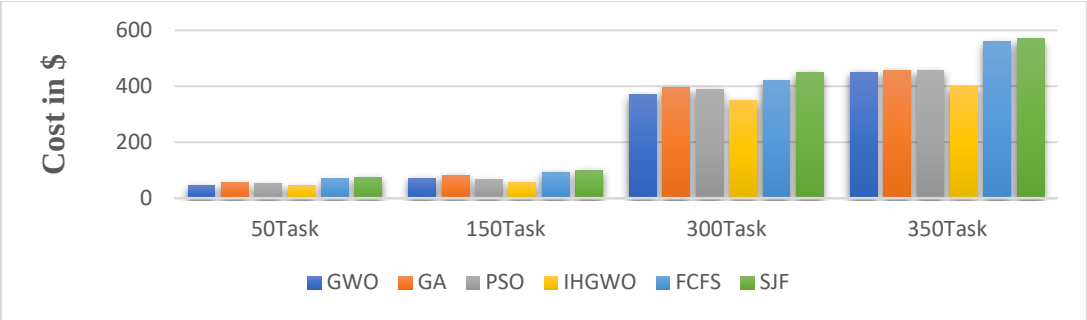

Figure 4. 6 Cost evaluation at aggregate(upper-level) fog node

Cost evaluation at cloud node

At the cloud node, we compared the suggested algorithm to various meta-heuristics algorithms with varying datasets in terms of costs without conflicting deadlines. The improved hybrid grey wolf optimization (IHGWO) algorithm was found to be more successful than other approaches, as shown in the figure below.

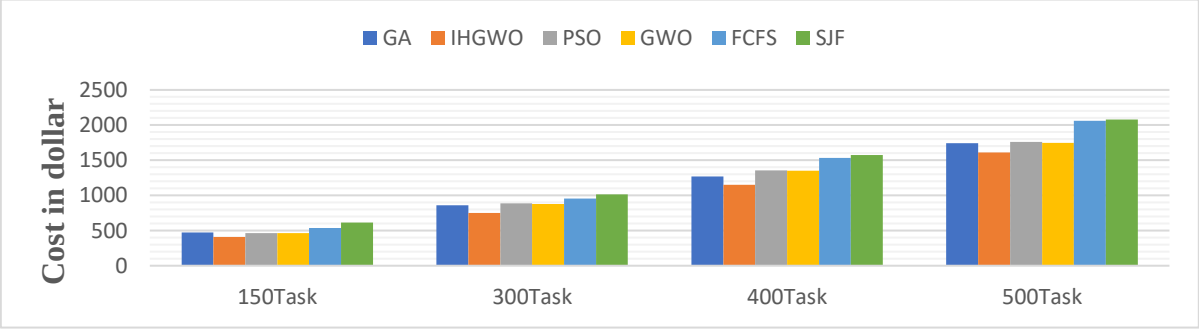

Figure 4. 7 Cost evaluation at cloud node

The proposed algorithm evaluation result summary

The effectiveness of the proposed algorithm has been evaluated using multiple metrics at the same time, such as cost and latency, without interfering with task deadlines. Based on the experiment results, the proposed algorithm has reduced cost by 17.67% and latency by 22.45% compared to FCFS, and has reduced cost by 18% and latency by 21.9% compared to SJF. It has

also reduced latency by 4% and cost reduction by 7.1%, outperforming PSO, reduced latency by 4.4% and cost reduction by 6.7%GWO, and reduced latency by 5% and cost reduction by 6.65%, outperforming GA. Furthermore, when compared to existing algorithms, the proposed approach more successfully meets the requirement of a task deadline.

Table 4. 2 result summary

Name of Algorithms	**parameters**	
	cost	**latency**
GA	6.65%	5%
PSO	7.1%	4%
GWO	6.7%	5%
FCFS	17.67%	22.45%
SJF	18%	21.9%

The proposed algorithm as we mentioned in the above table and we have discussed has reduced by the percent listed in terms of cost and latency without conflicting deadlines of tasks.

Chapter summary

In this research work, the performance improved hybrid grey wolf optimization algorithm has been evaluated with three other meta-heuristic algorithms like particle swarm optimization, grey wolf optimization, and genetic algorithm. In addition, first come first serve and short job first have been evaluated with the proposed algorithm in terms of latency reduction, cost reduction, and resource utilization with respect to task deadline with various datasets. At layers of nodes, such as low-level (nearest) fog nodes, upper-level (aggregate) fog nodes, and cloud nodes, the cost and latency were assessed. As it can be noticed from bar charts, the proposed algorithm reduces latency and cost by distributing the task to the low-level(nearest) fog, aggregate(upper-level), and cloud node based on the deadline better than other approaches.

On the cloudsim simulator, the effectiveness of the proposed approach was evaluated and compared to First Come First Served (FCFS), Short Job First (SJF), Particle Swarm Optimization (PSO), Grey Wolf Optimizer (GWO), and Genetic Algorithm (GA) As a result, we found that our algorithm outperformed the previously mentioned ones in terms of latency, reduced costs, and meeting task deadlines.

CHAPTER FIVE

5. Conclusion and Future Work

This chapter summarizes the research work and additional work that needs to be done in the future to make fog computing environments provide quality services to new emerging latency-sensitive tasks.

5.1. Conclusion

In this research, an improved hybrid grey wolf optimization algorithm has been proposed to meet the deadline, reduce cost and latency of a sequence of tasks executed on a virtual machine for fog-cloud computing. Fog computing is an emerging technology for data processing that is assumed as a solution to the latency problem arising from IoT devices. Though fog devices can be taken as a possible solution, tasks executed on each device have to be scheduled in a manner that reduces cost and latency with deliberate task deadline.

Task scheduling is an NP-hard problem that requires a meta-heuristics algorithm to get optimal solutions. In this research, we proposed an improved hybrid grey wolf optimization algorithm to balance the load on fog-cloud nodes. For this reason, we classify tasks into small, medium, and big size tasks based on the threshold values, as well as latency-sensitive, latency-tolerant(less-sensitive), and latency-insensitive. Small and latency-sensitive tasks should be executed on the low-level (nearest) fog node, while medium and latency-tolerant tasks should be executed on the upper-level (aggregate) fog node. Large and latency-insensitive tasks should be executed on cloud nodes.

The proposed algorithm was compared using the cloudsim simulator to well-known algorithms such as particle swarm optimization (PSO), genetic algorithm (GA), grey wolf optimization (GWO), first come first served (FCFS), and short job first (SJF). Based on experimental results, we found that, when the task deadline is taken into account, the proposed algorithm reduces cost and latency more than other specified algorithms.

5.2. Contribution

Cost reduction

Users only pay for the time they actually use the virtual machine; the provider sets the cost. The proposed approach reduced costs by assigning nodes tasks that, if achieved by the deadline,

would have the lowest costs, as well as by adjusting the task's cost and latency while taking user preferences into account.

Latency minimizes

The proposed method reduces task latency by allocating tasks to various levels, such as low-level (nearest) fog nodes, upper-level (aggregate) fog nodes, and cloud nodes in accordance with their deadline and task size. Additionally, the suggested algorithm minimizes task delay generally while maintaining user preferences for cost and latency without going against task deadlines.

Deadline achievement based on requirement

Since the task is divided into latency-sensitive, latency-tolerant(less-sensitive), and latency-insensitive classifications as well as small, medium, and large task sizes based on a given threshold, the proposed approach has successfully met the application deadline based on comparisons with other approaches. To find efficient nodes and meet the deadline, the proposed approach algorithm has a better nodes selection technique.

5.3. Future work

Even though the more efficient quality of service such as latency reduction, cost reduction with the satisfied deadline has been achieved by the proposed algorithm, still there is a gap as listed as follows.

Security Issues During Migration

Since different fog nodes involve in the fog layer, sharing of task among nodes may involve malicious fog nodes which compromise security of the data.

Fault tolerance issues

When a task is offloaded to another layer or node due to capacity constraints, the task may expire if the deadline has passed. As a result, it is necessary to deal with such a problem by improving the service quality. Concepts for fault tolerance

Dependent Task (Workflow)

Some tasks can be dependent on other task’s execution to complete their execution. Therefore, as a future idea, workflow scheduling has to be well studied to achieve better resource utilization.

Nature-inspired algorithm with minimum time complexity

Nature-inspired algorithms that take low time complexity for execution have to be considered in the future for assigning the task to the efficient node at the expected time.

Special Acknowledgment

This research project is funded by Adama Science and Technology University under the grant number:

ASTU/SM-R/ 490/22

Adama, Ethiopia

Reference

Ababneh, J. (2021). *A Hybrid Approach Based on Grey Wolf and Whale Optimization Algorithms for Solving Cloud Task Scheduling Problem*. *2021*.

Ai, Y., Peng, M., & Zhang, K. (2017). Wireless Communications for Ministry of Education , Beijing University of Posts and Author Information : *Digital Communications and Networks*. https://doi.org/10.1016/j.dcan.2017.07.001

Aladwani, T. (n.d.). *Types of Task Scheduling Algorithms in Cloud Computing Environment*. 1–12.

Alizadeh, M. R., Khajehvand, V., Rahmani, A. M., & Akbari, E. (2020). Task scheduling approaches in fog computing: A systematic review. *International Journal of Communication Systems*, *33*(16), 1–36. https://doi.org/10.1002/dac.4583

Alsadie, D., & Member, G. S. (2021). *TSMGWO : Optimizing Task Schedule Using Multi-Objectives Grey Wolf Optimizer for Cloud Data Centers*. *9*, 37707–37725.

Alzaqebah, A., Al-Sayyed, R., & Masadeh, R. (2019). Task scheduling based on modified grey Wolf optimizer in cloud computing environment. *2019 2nd International Conference on New Trends in Computing Sciences, ICTCS 2019 - Proceedings*, 1–6. https://doi.org/10.1109/ICTCS.2019.8923071

Bahwaireth, K., Tawalbeh, L., Benkhelifa, E., Jararweh, Y., & Tawalbeh, M. A. (2016). Experimental comparison of simulation tools for efficient cloud and mobile cloud computing applications. *EURASIP Journal on Information Security*. https://doi.org/10.1186/s13635-016-0039-y

Barik, R. K. (2017). *FogGrid : Leveraging Fog Computing for Enhanced Smart Grid Network*.

Bitam, S., Zeadally, S., & Mellouk, A. (2018). Fog computing job scheduling optimization based on bees swarm. *Enterprise Information Systems*, *12*(4), 373–397. https://doi.org/10.1080/17517575.2017.1304579

Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012). Fog computing and its role in the internet of things. *MCC'12 - Proceedings of the 1st ACM Mobile Cloud Computing Workshop*, 13–15. https://doi.org/10.1145/2342509.2342513

Byers, C. C. (2017). *Architectural Imperatives for Fog Computing : Use Cases , Requirements , and Architectural Techniques for Fog-Enabled IoT Networks*. *August*, 14–20.

Calheiros, R. N., Ranjan, R., Beloglazov, A., & Rose, A. F. De. (2011). *CloudSim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms*. *August 2010*, 23–50. https://doi.org/10.1002/spe

Carbas, S., Toktas, A., & Ustun, D. (n.d.). *Nature-Inspired Metaheuristic Algorithms for Engineering Optimization Applications*.

Chakraborty, A., & Kar, A. K. (2017). Swarm intelligence: A review of algorithms. *Modeling and Optimization in Science and Technologies*, *10*(March), 475–494. https://doi.org/10.1007/978-3-319-50920-4_19

cloudsimsim.pdf. (n.d.).

Endo, P. T., Rodrigues, M., Gonçalves, G. E., Kelner, J., & Sadok, D. H. (2016). *High availability in clouds : systematic review and research challenges*. https://doi.org/10.1186/s13677-016-0066-8

February, U., Internet, T., Revolution, F. I., Things, M., Things, M., Cities, S., & Challenge, S. C. (2020). *The Internet of Things (IoT): An Overview*. *2019*.

Ghobaei-Arani, M., Souri, A., & Rahmanian, A. A. (2020). Resource Management Approaches in Fog Computing: a Comprehensive Review. *Journal of Grid Computing*, *18*(1), 1–42. https://doi.org/10.1007/s10723-019-09491-1

Gupta, H., & Buyya, R. (2017). *iFogSim : A toolkit for modeling and simulation of resource management techniques in the Internet of Things , Edge*. *May*, 1275–1296. https://doi.org/10.1002/spe.2509

Hafdi, K., Kriouile, A., & Kriouile, A. (2019). *Overview on Internet of Things (IoT) Architectures , Enabling Technologies and Challenges*. *14*(9), 557–570. https://doi.org/10.17706/jcp.14.9

Hao, Z., Novak, E., Yi, S., & Li, Q. (2017). Challenges and Software Architecture for Fog Computing. *IEEE Internet Computing*, *21*(2), 44–53. https://doi.org/10.1109/MIC.2017.26

Hassan, R. (2004). Particle Swarm Optimization: Method and Applications. *Engineering*, 1–40.

Hoang, D., & Dang, T. D. (2017). FBRC: Optimization of task scheduling in fog-based region and cloud. *Proceedings - 16th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 11th IEEE International Conference on Big*

Data Science and Engineering and 14th IEEE International Conference on Embedded Software and Systems, 1109–1114. https://doi.org/10.1109/Trustcom/BigDataSE/ICESS.2017.360

Hosseinioun, P., Kheirabadi, M., Kamel Tabbakh, S. R., & Ghaemi, R. (2020a). A new energy-aware tasks scheduling approach in fog computing using hybrid meta-heuristic algorithm. *Journal of Parallel and Distributed Computing*, *143*, 88–96. https://doi.org/10.1016/j.jpdc.2020.04.008

Hosseinioun, P., Kheirabadi, M., Kamel Tabbakh, S. R., & Ghaemi, R. (2020b). aTask scheduling approaches in fog computing: A survey. *Transactions on Emerging Telecommunications Technologies*, *May 2019*, 1–11. https://doi.org/10.1002/ett.3792

Hu, P., Dhelim, S., Ning, H., & Qiu, T. (2017). Author ' s Accepted Manuscript Technologies , Applications and Open Issues Reference : *Journal of Network and Computer Applications*. https://doi.org/10.1016/j.jnca.2017.09.002

Huang, R., Sun, Y., Huang, C., Zhao, G., & Ma, Y. (2019). A Survey on Fog Computing. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Vol. 11637 LNCS*. Springer International Publishing. https://doi.org/10.1007/978-3-030-24900-7_13

Humane, P., & Varshapriya, P. J. N. (2015). *Simulation of Cloud Infrastructure using CloudSim Simulator : A Practical Approach for Researchers*. *May*, 207–211.

IDC. (2016). *IDC: The premier global market intelligence firm.* http://www.idc.com/

Juneja, M., & Nagar, S. K. (2016). *Particle swarm optimization algorithm and its parameters : A review*. *Iccccm*.

Kabirzadeh, S., Rahbari, D., & Nickray, M. (2018). A hyper heuristic algorithm for scheduling of fog networks. *Conference of Open Innovation Association, FRUCT*, 148–155. https://doi.org/10.23919/FRUCT.2017.8250177

Khanvilkar, S., Bashir, F., Schonfeld, D., & Khokhar, A. (n.d.). Multimedia Networks and Communication. In *The Electrical Engineering Handbook*. Elsevier Inc. https://doi.org/10.1016/B978-012170960-0/50033-5

Klas, G. I. (2015). *Fog Computing and Mobile Edge Cloud Gain Momentum Open Fog Consortium , ETSI MEC and Cloudlets*. 1–13.

Kumar, R. (n.d.). *Mobile cloud computing*.

Li-wenGuo, Z. T. J. L. (2019). An improved hybrid grey wolf optimization algorithm. *Soft Computing*, *23*(15), 6617–6631. https://doi.org/10.1007/s00500-018-3310-y

Liu, Y., Fieldsend, J. E., & Min, G. (2017a). A framework of fog computing: Architecture, challenges, and optimization. *IEEE Access*, *5*, 25445–25454. https://doi.org/10.1109/ACCESS.2017.2766923

Liu, Y., Fieldsend, J. E., & Min, G. (2017b). SPECIAL SECTION ON CYBER-PHYSICAL-SOCIAL COMPUTING AND NETWORKING A Framework of Fog Computing: Architecture, Challenges, and Optimization. *IEEE Access*, *5*, 25445–25454.

Mahmud, R., Kotagiri, R., & Buyya, R. (2018). Fog Computing: A taxonomy, survey and future directions. *Internet of Things*, *0*(9789811058608), 103–130. https://doi.org/10.1007/978-981-10-5861-5_5

Mao, Y., You, C., Zhang, J., Huang, K., & Letaief, K. B. (n.d.). *A Survey on Mobile Edge Computing : The Communication Perspective*. 1–37.

Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey Wolf Optimizer. *Advances in Engineering Software*, *69*, 46–61. https://doi.org/10.1016/j.advengsoft.2013.12.007

Moghaddam, F. F., Rohani, M. B., Ahmadi, M., Khodadadi, T., & Madadipouya, K. (2015). *Cloud Computing : Vision , Architecture and Characteristics*. 1–6.

Moravcik, M., Segec, P., & Kontsek, M. (2018). Overview of Cloud Computing standards. *2018 16th International Conference on Emerging ELearning Technologies and Applications (ICETA)*, 395–402.

Naha, R. K., Garg, S., Georgakopoulos, D., Jayaraman, P. P., Gao, L., Xiang, Y., & Ranjan, R. (2018a). Fog Computing : Survey of Trends ,. *IEEE Access*, *PP*(c), 1. https://doi.org/10.1109/ACCESS.2018.2866491

Naha, R. K., Garg, S., Georgakopoulos, D., Jayaraman, P. P., Gao, L., Xiang, Y., & Ranjan, R. (2018b). Fog computing: Survey of trends, architectures, requirements, and research directions. *IEEE Access*, *6*(c), 47980–48009. https://doi.org/10.1109/ACCESS.2018.2866491

Natesan, G., & Chokkalingam, A. (2020). An improved grey wolf optimization algorithm based task scheduling in cloud computing environment. *International Arab Journal of Information Technology*, *17*(1), 73–81. https://doi.org/10.34028/iajit/17/1/9

Negash, B., Rahmani, A. M., Liljeberg, P., & Jantsch, A. (1999). *Fog Computing*

Fundamentals in the Internet-of-Things. 3–13. https://doi.org/10.1007/978-3-319-57639-8

Nguyen, B. M., Binh, H. T. T., Anh, T. T., & Son, D. B. (2019). Evolutionary algorithms to optimize task scheduling problem for the IoT based Bag-of-Tasks application in Cloud-Fog computing environment. *Applied Sciences (Switzerland)*, *9*(9). https://doi.org/10.3390/app9091730

Nikoui, T. S., Balador, A., Rahmani, A. M., & Bakhshi, Z. (2020). Cost-Aware Task Scheduling in Fog-Cloud Environment. *Proceedings of RTEST 2020 - 3rd CSI/CPSSI International Symposium on Real-Time and Embedded Systems and Technologies*. https://doi.org/10.1109/RTEST49666.2020.9140118

Parikh, S. (2013). *Double Level Priority based Optimization Algorithm for Task Scheduling in Cloud Computing*. *62*(20), 33–37.

Rahman, G., & Wen, C. C. (2018). *Fog Computing , Applications , Security and Challenges , Review*. *7*(3), 1615–1621. https://doi.org/10.14419/ijet.v7i3.12612

Sabireen, H., & Neelanarayanan, V. (2021). A Review on Fog Computing : Architecture , Fog with IoT , Algorithms and Research Challenges. *ICT Express*, *7*(2), 162–176. https://doi.org/10.1016/j.icte.2021.05.004

Saurez, E., Gupta, H., & Mayer, R. (2017). *Demo Abstract : Fog Computing for Improving User Application Interaction and Context Awareness*. 285–286.

Science, A. A., & Teshome, T. (2019). *Design and Development of Task Scheduling Algorithm for Fog Computing Environment*. *October*.

Science, A. A., & Tsega, A. (2020). *ADDIS ABABA SCIENCE AND TECHNOLOGY UNIVERSITY DESIGNING A DEADLINE AWARE DATA OFFLOADING ALGORITHM FOR FOG COMPUTING ENVIRONMENT A Thesis Submitted as a Partial Fulfillment to the Requirements for the*. *June*.

Sharma, S., & Saini, H. (2019). ur na of. *Sustainable Computing: Informatics and Systems*, 100355. https://doi.org/10.1016/j.suscom.2019.100355

Sreenu, K., & Sreelatha, M. (2019). W-Scheduler: whale optimization for task scheduling in cloud computing. *Cluster Computing*, *22*(s1), 1087–1098. https://doi.org/10.1007/s10586-017-1055-5

Steps, N. (n.d.). *Edge Computing*.

Stream, C. (n.d.). *Challenges and Opportunities in Edge*. 59–70.

Sudheer, M. S., & Vamsi Krishna, M. (2019). Dynamic PSO for task scheduling optimization in cloud computing. *International Journal of Recent Technology and Engineering*, *8*(2 Special Issue 11), 332–338. https://doi.org/10.35940/ijrte.B1052.0982S1119

Survey, A. C. (2020). *SS symmetry Internet of Things and Its Applications :* 1–29.

Suryateja, P. S. (2016). *A Comparative Analysis of Cloud Simulators*. *July*. https://doi.org/10.5815/ijmecs.2016.04.08

Vuyyuru, M., Annapurna, P., Babu, K. G., & Ratnam, A. S. K. (2012). *An Overview of Cloud Computing Technology*. *3*, 244–246.

Wang, D., Tan, D., & Liu, L. (2018). Particle swarm optimization algorithm: an overview. *Soft Computing*, *22*(2), 387–408. https://doi.org/10.1007/s00500-016-2474-6

Wang, J., & Li, D. (2019). Task scheduling based on a hybrid heuristic algorithm for smart production line with fog computing. *Sensors (Switzerland)*, *19*(5). https://doi.org/10.3390/s19051023

Yi, S., Hao, Z., Qin, Z., & Li, Q. (2015). *Fog Computing : Platform and Applications*. https://doi.org/10.1109/HotWeb.2015.22

Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J., & Jue, J. P. (2019). *All one needs to know about fog computing and related edge computing paradigms : A complete survey ☆ , ☆☆*. *98*(December 2018), 289–330.

Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J., Jue, J. P., Fung, C., Kadiyala, K., & Dallas, U. T. (2019). PT. *Journal of Systems Architecture*. https://doi.org/10.1016/j.sysarc.2019.02.009

Yousefpour, A., Ishigaki, G., & Jue, J. P. (2017). Fog Computing: Towards Minimizing Delay in the Internet of Things. *Proceedings - 2017 IEEE 1st International Conference on Edge Computing, EDGE 2017*, 17–24. https://doi.org/10.1109/IEEE.EDGE.2017.12

Zhong, Z., Chen, K., Zhai, X., & Zhou, S. (2016). *Virtual Machine-Based Task Scheduling Algorithm in a Cloud Computing Environment*. *21*(6), 660–667.

Appendix

Tasks Assigned to Low-Level Fog Node in Appendix-I

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Information of Task Executed on Nearest or low-level Fog Node ~~~~~~

ID of Task	Arrival time of Task	Deadline of Task	Response time of Task	Node assigned(VM ID)	Cost
0	1.21	2.1	1.82	0	0.06
1	1.32	2.4	2.36	0	0.06
2	1.43	2.7	2.43	0	0.02
3	1.54	3	2.54	0	0.02
4	1.65	3.3	2.67	0	0.03
5	1.76	3.6	3.27	0	0.07
6	1.87	3.9	3.56	0	0.04
7	1.98	4.2	3.69	0	0.03
8	2.09	4.5	4.41	0	0.09
9	2.2	4.8	4.48	0	0.03
10	2.31	5.1	4.51	0	0.02
11	2.42	5.4	4.61	0	0.03
12	2.53	5.7	4.69	0	0.03
13	2.64	6	4.81	0	0.04
14	2.75	6.3	5.49	0	0.09
15	2.86	6.6	6.09	0	0.09
16	2.97	6.9	6.11	0	0.03
17	3.08	7.2	7.02	0	0.12
18	3.19	7.5	4.52	1	0.13
19	3.3	7.8	7.67	0	0.09
20	3.41	8.1	7.8	0	0.04
21	3.52	8.4	5.26	1	0.1
22	3.63	8.7	8.16	0	0.07
23	3.74	9	8.28	0	0.05
24	3.85	9.3	8.94	0	0.1
25	3.96	9.6	9.44	0	0.09
26	4.07	9.9	6.27	1	0.13
27	4.18	10.2	9.8	0	0.07
28	4.29	10.5	10.28	0	0.09
29	4.4	10.8	10.4	0	0.05
30	4.51	11.1	7.08	1	0.12
31	4.62	11.4	9.61	4	0.8

Tasks Assigned to upper-level fog node in Appendix-II

~~~~~~~~~~~~~~~~~~~~~~~~ Information of Task Executed on Aggregate or Upper-level Fog Node~~~~~~

ID of Task	Arrival time of Task	Deadline of Task	Response time of Task	Node assigned(VM ID)	Cost
350	7.01	23.41	23.19	40	1.15
349	7	23.38	8.5	46	0.29
348	6.99	23.35	23.17	39	0.73
347	6.98	23.32	8.43	46	0.42
346	6.97	23.29	23.26	38	0.7
345	6.96	23.26	23.16	37	0.68
344	6.95	23.23	23.21	43	3.52
343	6.94	23.2	23.09	45	3.52
342	6.93	23.17	23.17	36	0.62
341	6.92	23.14	8.3	46	0.36
340	6.91	23.11	8.19	46	0.34
339	6.9	23.08	22.94	38	0.64
338	6.89	23.05	22.86	44	3.39
337	6.88	23.02	8.09	46	0.47
336	6.87	22.99	22.9	36	0.63
335	6.86	22.96	7.94	46	0.36
334	6.85	22.93	22.88	40	1.21
333	6.84	22.9	22.83	39	0.77
332	6.83	22.87	22.85	37	0.64
331	6.82	22.84	7.84	46	0.46
330	6.81	22.81	7.69	46	0.49
329	6.8	22.78	22.38	41	1.21
328	6.79	22.75	22.61	42	3.5
327	6.78	22.72	22.04	41	1.14
326	6.77	22.69	22.45	39	0.66
325	6.76	22.66	22.64	38	0.59
324	6.75	22.63	22.62	36	0.65
323	6.74	22.6	7.54	46	0.37
322	6.73	22.57	22.56	37	0.58
321	6.72	22.54	22.53	40	3.24
320	6.71	22.51	22.38	38	0.65
319	6.7	22.48	22.14	39	1.21

Assignment of Tasks to Cloud Node in Appendix-III

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Information of Task Executed on Cloud Fog Node ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

ID of Task	Arrival time of Task	Deadline of Task	Response time of Task	Node assigned(VM ID)	Cost
351	100.9	140	102.78	36	2.72
352	101.9	141.5	103.43	35	0.61
353	102.9	143	103.21	36	0.45
354	103.9	144.5	103.92	47	0.06
355	104.9	146	106.24	37	1.94
356	105.9	147.5	106.04	46	0.37
357	106.9	149	107.01	49	0.3
358	107.9	150.5	108.05	49	0.39
359	108.9	152	110.43	43	3.75
360	109.9	153.5	112.74	37	4.11
361	110.9	155	114.72	37	5.53
362	111.9	156.5	112.03	46	0.35
363	112.9	158	114.66	40	4.3
364	113.9	159.5	116.26	39	3.43
365	114.9	161	115	46	0.27
366	115.9	162.5	117.23	42	3.25
367	116.9	164	127.94	35	4.42
368	117.9	165.5	119.34	38	2.09
369	118.9	167	120.2	44	3.18
370	119.9	168.5	121.34	42	3.53
371	120.9	170	122.71	44	4.43
372	121.9	171.5	123.17	38	1.84
373	122.9	173	123.43	39	0.76
374	123.9	174.5	124.32	36	0.61
375	124.9	176	126.73	35	0.73
376	125.9	177.5	127.07	39	1.7
377	126.9	179	127.34	37	0.63
378	127.9	180.5	128.93	44	2.52
379	128.9	182	129.07	47	0.44
380	129.9	183.5	132.42	39	3.66
381	130.9	185	133.08	35	0.87
382	131.9	186.5	132.09	48	0.5

Task Classification Model sample code -Appendix-IV

```
public void GreyWolf () {
        int [] fogdevice=new int [15];
        int [] clouddevice=new int [35];
        int fog_number=20;
        int cloud_number=30;
        double rohwfog, rowhcloud, probablityfog, probablitycloud;
                Scanner scanner;
         int [] tall = new int [500];
            int a = 0, f=0, c=0;
            double num_fog=0;
        try {
    scanner = new Scanner (new File("C://Users//Gado//Desktop//GoCJ_Dataset_500.txt"));
                        while (scanner. hasNextInt ()) {
                            if (fogdevice. length==0) {
                    fog_arrival_rate=scanner. nextInt ();
                } else {
                        fog_arrival_rate=sum_array(fogdevice)/fogdevice. length;
                        num_fog=sum_array(fogdevice)/(20*fog_service_rate);    }
                                    if (clouddevice. length==0) {
                    cloud_arrival_rate=scanner. nextInt ();
                } else {
                    cloud_arrival_rate=sum_array(clouddevice)/clouddevice. length;        }
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

```
                    rohwfog=fog_arrival_rate/fog_service_rate;
                rowhcloud=cloud_arrival_rate/cloud_service_rate;
                probablityfog=Probability (rohwfog, fog_number);
                probablitycloud=Probability (rowhcloud, cloud_number);
                tall[a++] = scanner. nextInt ();                                    }
        } catch (FileNotFoundException ex) {
        }
                                    @SuppressWarnings("unused")
                    Datacenter datacenter0 = createDatacenter("Datacenter_0");
                    @SuppressWarnings("unused")
                    Datacenter datacenter1 = createDatacenter("Datacenter_1");
                                        DatacenterBroker broker = createBroker ();
                    int brokerId = broker. getId ();
                                    cloudletList = createCloudlet (brokerId, 500, tall);
                vmlist = createVM (brokerId, 50);
             broker. submitVmList(vmlist);
            broker. submitCloudletList(cloudletList);
            CloudSim.startSimulation();
                    double taskdeadline;
                    double inputfilesize;
                for (int i = 0; i< cloudletList.size(); i++) {
                    taskdeadline=Task_Info[i][4]-Task_Info[i][3];
                     inputfilesize=Task_Info[i][1] +Task_Info[i][2];
                       if(taskdeadline<5.3&&inputfilesize<=100) {
                            delay_sensitive.add(i); }
            if(taskdeadline>5.3 && taskdeadline<20.83&&inputfilesize<=200) {
                                delay_tolerant.add(i);
                    } else if(taskdeadline>20.83||inputfilesize>200) {
                                delay_sensitive.add(i); }    }
```

Sample code calculate and sorting the fitness

```
public void SortingValues () {
        double latency_sum, current_latency, cldletLength;
        int Vm_cap, vm_cap_id;
        double cost, current_cost, makespan, vm_cap2, TaskLength, inputfilesize,
outputfilesize, bandwidth;
          int currentcost=0, j=0, taskid, index;
         Vm vm_current;
        Cloudlet cldlet;
    for (int i = 0; i < num_solution; i++) {
            current_cost=0;
             latency_sum=0.0;
                        makespan=0.0;
                    for (int f = 0; f < delay_insensitive. size (); f++) {
```

```
vm_cap_id=task_assigned[i][f];
index= (Integer) delay_insensitive.get(f);
TaskLength=Task_Info[index][0];
inputfilesize=Task_Info[index][1];
outputfilesize=Task_Info[index][2];
vm_cap2 = vm_data[vm_cap_id][0] ;//vm capacity
cost = vm_data[vm_cap_id][2] ;//cost
bandwidth=vm_data[vm_cap_id][1] ;// bandwidth
makespan=Double.valueOf(dcformat.format(((TaskLength/(double)vm_cap2))))+makespan;
current_cost=Double.valueOf(dcformat.format((TaskLength/(double)vm_cap2)*cost))+(inputf
ilesize+outputfilesize/(double)bandwidth)*cost;
current_latency=Double.valueOf(dcformat.format((alpha*((TaskLength/(double)vm_cap2))
+((inputfilesize+outputfilesize/(double)bandwidth))) +(1-alpha)*( current_cost)));
latency_sum=Double.valueOf(dcformat. format(current_latency));  }
fitness[i]=latency_sum;
costfitness[i]=current_cost;
makespanarr[i]=makespan;}}
```

Latency sensitive sample code Appendix-V

```
int value=0, index = 0;
double tasklen, inputsize, outputsize, deadline, mips, costtotal = 0;
double avg_wt=0.004, cost=0.0;
double bandwidth, costmin=0.0, ServiceDelay=0.0;
double [][] TaskStatusInformation=new double [delay_sensitive. size ()][6];
double [][] TaskStatusInformationaggregate=new double [delay_tolerant. size ()][6];
int j=0;
double responsetimetotal=0, costspenttotal=0;
while (delay_sensitive. size ()>0)      //   vm_id=task_assigned [0] [optimum];   {
boolean assigned=false;
value=0;
index=(Integer)delay_sensitive.get(value);
taskdeadline=Task_Info[index][4]-Task_Info[index][3];
System.out.println(" let see index "+index);
System.out.println(" deadline "+taskdeadline);
tasklen=Task_Info[index][0];
inputsize=Task_Info[index][1];
outputsize=Task_Info[index][2];
deadline=Task_Info[index][4];
double servdelay1;
for (j=0; j<=10; j++) {
mips=vm_data[j][0];
bandwidth=vm_data[j][1];
cost=vm_data[j][2];
avg_wt=NodeController[1][j];
```

```
                    if (Task_Info[index][3]>avg_wt) {
                        avg_wt=0;
                    } else {
                        avg_wt=avg_wt-Task_Info[index][3]; }
        ServiceDelay=Task_Info[index][3]+avg_wt+tasklen/(double)(mips)+(inputsize+output
size)/(double)bandwidth;
                    if (ServiceDelay<deadline) {
                        assigned=true;
                        boolean exist=false;
NodeController[1][j]=Task_Info[index][3]+avg_wt+tasklen/(double)(mips);
costnode=((tasklen/(mips))*(cost))+(((inputsize+outputsize)/(double)bandwidth)*(cost));
                    task_assigned_inf [0] [index]=index;
                        task_assigned_inf [1] [index]=j;
                        task_assigned_inf [2] [index]=costnode;
                        delay_sensitive. remove(value);
                    break;}}
            if (! assigned) {
                for (j=11; j<=35; j++) {
                    mips=vm_data[j][0];
                    bandwidth=vm_data[j][1];
                    cost=vm_data[j][2];
                    avg_wt=NodeController[1][j];
                    if (Task_Info[index][3]>avg_wt) {
                        avg_wt=0;
                    } else {
                        avg_wt=avg_wt-Task_Info[index][3]; }
        ServiceDelay=Task_Info[index][3]
+avg_wt+tasklen/(double)(mips)+(inputsize+outputsize)/(double)bandwidth;
                    if (ServiceDelay<deadline) {
                        assigned=true;
  costnode=((tasklen/(mips)) *(cost)) +(((inputsize+outputsize)/(double)bandwidth) *(cost));
          NodeController[1][j] =Task_Info[index][3] +avg_wt tasklen/(double)(mips);
                        task_assigned_inf [0] [index]=index;
                        task_assigned_inf [1] [index]=j;
                        task_assigned_inf [2] [index]=costnode;
                        delay_sensitive. remove(value);
                        break;}}}
```

latency less-sensitive sample code Appendix-VI

```
            double responsetimetotal1=0, costspenttotal1=0;
            while (delay_tolerant. size ()>0) {
                value=0;
                boolean assigned=false;
                index=(Integer)(delay_tolerant.get(value));
```

```
                double      ServiceDelay1=0.0, ServiceDelay2=0.0;
                        tasklen=Task_Info[index][0];
                        inputsize=Task_Info[index][1];
                        outputsize=Task_Info[index][2];
                        deadline=Task_Info[index][4];
                        int efficentindex=11;
                        int entered=0;//check for assignment
                        for (j=11; j<=35; j++) {
                                mips=vm_data[j][0];
                                bandwidth=vm_data[j][1];
                                cost=vm_data[j][2];
                                avg_wt=NodeController[1][j];
                                if (Task_Info[index][3]>avg_wt) {
                                        avg_wt=0;
                                } else {
                                        avg_wt=avg_wt-Task_Info[index][3]; }
                                ServiceDelay=Task_Info[index][3]
          +avg_wt+tasklen/(double)(mips)+(inputsize+outputsize)/(double)bandwidth;
                if(assigned) {
                bandwidth=vm_data[efficentindex][1];
                cost=vm_data[efficentindex][2];
 NodeController [1] [efficentindex]=Task_Info[efficentindex][3] +av+tasklen/(double)(mips);
costtotal=((tasklen/(vm_cap2)) *(cost1)) +(((inputsize+outputsize)/(double)bandwidth)
*(cost));
                task_assigned_inf [0] [index]=index;
                task_assigned_inf [1] [index]=efficentindex;
                task_assigned_inf [2] [index]=costnode;
                delay_tolerant. remove(value);          }
             if (! assigned) {
                for (j=36; j<=50; j++) {
                                mips=vm_data[j][0];
                                bandwidth=vm_data[j][1];
                                cost=vm_data[j][2];
                                avg_wt=NodeController[1][j];
                                if (Task_Info[index][3]>avg_wt) {
                                        avg_wt=0;
                                } else {
                                        avg_wt=avg_wt-Task_Info[index][3]; }
         ServiceDelay2=Task_Info[index][3]
+avg_wt+tasklen/(double)(mips)+(inputsize+outputsize)/(double)bandwidth;
                                if (ServiceDelay2<deadline) {
                                        ServiceDelay=ServiceDelay2;
                                                mincost=mipsnode;
```

```
                    exist=true; }}
                if (! exist) {
                mincost=mipsnode; }
        NodeController[1][j] =Task_Info[index][3] +avg_wt tasklen/(double)(mips);
    costtotal=((tasklen/(vm_cap2)) *(cost1)) +(((inputsize+outputsize)/(double)bandwidth)
                    *(cost));
                task_assigned_inf [0] [index]=index;
                efficentindex=j;
                task_assigned_inf [1] [index]=j;
                task_assigned_inf [3] [index]=costnode;
                delay_tolerant. remove(value);
                break; }}}
```

Sample code of Latency-insensitive Appendix-VII

```
int vm_id, index2;
        double tasklen2, inputsize2, outputsize2, deadline2, cost2, vm_cap2, avg_wt2;
        double [][] TaskStatusInformation2=new double [delay_insensitive. size ()][6];
                for (int optimum=0; optimum<delay_insensitive. size (); optimum++) {
          index2=(Integer)delay_insensitive.get(optimum);
                    tasklen=Task_Info[index2][0];
                    inputsize=Task_Info[index2][1];
                    outputsize=Task_Info[index2][2];
                    deadline=Task_Info[index2][4];
                  vm_id=task_assigned [0] [optimum];
                bandwidth=vm_data[vm_id][1];
                    cost=vm_data[vm_id][2];
                    vm_cap2 = vm_data[vm_id][0] ;//vm capacity
                    avg_wt=NodeController [1] [vm_id];
                            if (Task_Info[index2][3]>avg_wt) {
                    avg_wt=0;
                    } else {
                    avg_wt=avg_wt-Task_Info[index2][3]; }
ServiceDelay=Task_Info[index2][3]  +avg_wt  tasklen/(double)(vm_cap2)
+(inputsize+outputsize)/(double)bandwidth;
                    if (ServiceDelay<=deadline) {
        NodeController [1] [vm_id] =Task_Info[index][3] +avg_wt
tasklen/(double)(vm_cap2);
    costtotal=((tasklen/(vm_cap2)) *(cost1)) +(((inputsize+outputsize)/(double)bandwidth)
                    *(cost));
                    } else {
                    for (int s=35; s<=50; s++) {
                        index2=(Integer)delay_insensitive.get(optimum);
                        tasklen=Task_Info[index2][0];
                        inputsize=Task_Info[index2][1];
```

```
                                        outputsize=Task_Info[index2][2];
                                        deadline=Task_Info[index2][4];
                                        bandwidth=vm_data[s][1];
                                        cost=vm_data[s][2];
                                        vm_cap2 = vm_data[s][0] ;//vm capacity
                                avg_wt=NodeController[1][s];
                                                if (Task_Info[index][3]>avg_wt) {
                                                avg_wt=0;
                                        } else {
                                                avg_wt=avg_wt-Task_Info[index2][3];
                                        }
ServiceDelay=Task_Info[index2][3]          +avg_wt          tasklen/(double)(vm_cap2)
+(inputsize+outputsize)/(double)bandwidth;
        if (ServiceDelay<deadline) {
                                task_assigned [0] [optimum]=s;
                                        vm_id=s
                                mincost=mipsnode;
                        exist=true;}}
                                if (! exist) {
                                mincost=mipsnode;    }
  NodeController[1][s] =Task_Info[index2][3] +avg_wt tasklen/(double)(vm_cap2);
                                costtotal=((tasklen/(vm_cap2)) *(cost1))
                        +(((inputsize+outputsize)/(double)bandwidth) *(cost));
                                                                break;       }}}
```