Resource Optimization Techniques for IoT-Edge Environment

Thesis Submitted

By

Mohana Bakshi

Doctor of Philosophy (Engineering)

Department of Computer Science and Engineering

Faculty Council of Engineering & Technology

Jadavpur University Kolkata, India

2024

Index No. 284/18/E

1. **Title of the Thesis:** Resource Optimization Techniques for IoT-Edge Environment

2. **Name, Designation and Institution of the Supervisor:**

Dr. Chandreyee Chowdhury
Professor
Department of Computer Science and Engineering
Jadavpur University Kolkata,
West Bengal-700032, India

Dr. Ujjwal Maulik
Professor
Department of Computer Science and Engineering
Jadavpur University Kolkata,
West Bengal-700032, India

3. **List of Publications**

(a) Journal Publications:

Published

1. **M.Bakshi,** C.Chowdhury and U.Maulik, Energy-efficient cluster head selection algorithm for IoT using modified glow-worm swarm optimization," The Journal of Supercomputing, Springer, 2021, DOI: 77(7), pp. 6457-6475. (**IF** : 2.5)

   https://doi.org/10.1007/s11227-020-03536-z,

2. **M.Bakshi,** C.Chowdhury and U.Maulik, Cuckoo search optimization-based energy efficient job scheduling approach for IoT-edge environment," Springer, 2023, DOI: 79(16), pp. 18227-18255. (**IF** : 2.5)

   https://doi.org/10.1007/s11227-023-05358-1,

Communicated

1. M.Bakshi, U.Maulik and C.Chowdhury, Cuckoo Search based Deadline Aware Energy Optimized Task Offloading Strategy," communicated to SN Computer Science Journal, Springer.

(b) Conference Publications:

1. **M.Bakshi,** M.Roy, U.Maulik and C. Chowdhury, An optimal edge server placement algorithm based on glowworm swarm optimization technique ," in 4th International conference on frontiers in computing and systems, Mandi, Himachal Pradesh, 2023. https://doi.org/10.1007/978-981-97-2614-1_1

4. **List of Patents :** None

5. **List of Presentation in International Conference:**

1. **M.Bakshi,** M.Roy, U.Maulik and C.Chowdhury, An optimal edge server placement algorithm based on glowworm swarm optimization technique," in 4th International conference on frontiers in computing and systems, Mandi, Himachal Pradesh, 2023. https://doi.org/10.1007/978-981-97-2614-1_1

Statement of Originality

I, **Ms. Mohana Bakshi** registered on **25th May, 2018** do hereby declare that this thesis entitled **"Resource Optimization Techniques for IoT-Edge Environment"** contains literature survey and original research work done by the undersigned candidate as part of Doctoral studies.

All information in this thesis have been obtained and presented in accordance with existing academic rules and ethical conduct. I declare that, as required by these rules and conduct, I have fully cited and referred all materials and results that are not original to this work.

I also declare that I have checked this thesis as per the **"Policy on Anti Plagiarism, Jadavpur University, ~~2023~~ 2019"**, and the level of similarity as checked by iThenticate software is **8**% .

Signature of Candidate: Mohana Bakshi 11.08.2025

Date :

Certified by Supervisor(s):

(Signature with date, seal)

1. Chandreyee Chowdhury 11.08.2025

(Dr. Chandreyee Chowdhury,
Professor,
Department of Computer Science and Engineering
Jadavpur University.)

Professor
Computer Sc. & Engg. Department
Jadavpur University
Kolkata-700032

2. Ujjwal Maulik 11.08.2025

(Dr. Ujjwal Maulik,
Professor,
Department of Computer Science and Engineering
Jadavpur University.)

Professor
Computer Sc. & Engg. Department
Jadavpur University
Kolkata-700032

Certificate from the Supervisor

This is to certify that the thesis entitled ***"Resource Optimization Techniques for IoT-Edge Environment"*** *submitted by* ***Ms. Mohana Bakshi****, who got her name registered on* ***25th May 2018****, for the award of Ph.D.(Engineering) degree of Jadavpur University, is absolutely based upon her own work under the supervision of* ***Dr. Chandreyee Chowdhury*** *and* ***Dr. Ujjwal Maulik*** *and that neither her thesis nor any part of the thesis has been submitted for any degree/diploma or any other academic award anywhere before.*

Chandreyee Chowdhury
11-08-2025

Dr. Chandreyee Chowdhury,
Professor,
Department of Computer
Science and Engineering
Jadavpur University.
(Supervisor)

Professor
Computer Sc. & Engg. Department
Jadavpur University
Kolkata-700032

Ujjwal Maulik
11-08-2025

Dr. Ujjwal Maulik,
Professor,
Department of Computer
Science and Engineering
Jadavpur University.
(Supervisor)

Professor
Computer Sc. & Engg. Department
Jadavpur University
Kolkata-700032

Acknowledgment

Reflecting on my Ph.D. journey, I feel truly fortunate and appreciative for all that I've gained. I could not have completed my thesis without the support, guidance, and encouragement from many individuals who played essential roles along the way. I sincerely acknowledge their contributions with gratitude and joy.

First I would like to express my deepest gratitude and sincere thanks to my supervisors Dr. Chandreyee Chowdhury, Professor, Jadavpur University and Dr. Ujjwal Maulik, Professor, Jadavpur University for guiding me in all aspect to carry out and complete this research work towards partial fulfillment of Doctor of Philosophy. Without their enthusiasm, integral view on research, dedicated help, guidance, inspiration, constant feedback, continuous support, this PhD would not have been achievable.

I would like to sincerely thank to my other thesis committee members Dr. Sarmistha Neogy,Professor, Jadavpur University, Dr. Sarbani Roy, Professor, Jadavpur University, for providing insightful and constructive comments to improve the quality of this thesis. I would like to thank Computer Science and Engineering Department, Jadavpur University to give this opportunity and carrying my research with this excellent support and guidance.

I would like to express my heartfelt thanks to Asif Iqbal Midya, Assistant Professor (FIEM). His willingness to help as co-fellow whenever I needed it was truly invaluable throughout the course of my thesis. My special thanks to research fellows of Jadavpur University, CSE Department, Ayan, Moumita for their cooperation and continuous support.

I would like to express my sincere thanks to Mr. Abhijit Saha, Principal , PM SHRI Kendriya Vidyalaya Burdwan, for supporting and encouraging me to do the research. I am grateful to my students of PM SHRI KV Burdwan, as their enthusiasm and interest in computer science has always motivated me to do my research work.

Now, it is the time to express my gratitude and heartiest thanks to my family who always being my backbone and source of inspiration and motivation to work. I would like to show my sincere regards to my parents who always supported me, and encouraged me a lot to follow my dreams. I am very much thankful to my late father-in-law and late mother-in-law for inspiring me to reach my goal.

My heartfelt deepest thanks to my husband, Raja Goswami, for his continuous support and understanding of my goals and aspirations. He always stands as a pillar in my success, failure and problem. He managed all the problems, looked after our son to continue my research work. Without his cooperation, encouragement I may not be able to complete my thesis.

My son, Riyan, is always the biggest reason for my inspiration and happiness that helped me to work with pleasure. I feel whatever I learned throughout my Ph.D. life from all aspects is for him so that he could find his way with this knowledge.

Last, but not least, I am thankful to the almighty God for giving me life to live my dreams.

Dedicated

To my family

To all the researchers

Abstract

The rapid growth of the Internet of Things (IoT) has led to an increased demand for efficient data processing closer to the data source, making edge computing a critical component of modern IoT ecosystems. To ensure improved overall performance, task execution in an IoT-edge-cloud environment poses significant resource optimisation challenges. With an emphasis on methods to increase the performance efficiency of IoT-Edge environments, this thesis investigates several resource optimization approaches for IoT edge environments. The study begins by analyzing traditional resource optimization methods and their limitations in IoT-Edge environments. It identifies three critical components of job execution in such environment: (i) placement of edge servers, (ii) job offloading, and (iii) task scheduling strategies. The investigation of state-of-the-art techniques has revealed significant metrics still needing attention. The primary challenges to resource optimization in IoT-Edge environments include: (i) Need of strategies capable of determining the optimal number of edge servers while extending network lifespan and maximizing coverage. (ii) In resource optimization, many existing task-offloading techniques fail to account for minimizing task cancellation for time-sensitive jobs. (iii) Most job scheduling methods tend to focus on factors like QoS, migration costs, and resource utilization, overlooking critical elements such as job priority, conflicts, and dependencies that are vital for efficient job execution at edge nodes. This thesis addresses these challenges. The first challenge is tackled by introducing a method for selecting cluster heads using Glowworm Swarm Optimization, which effectively divides the network into an optimal number of clusters while balancing the needs of edge servers and associated base stations. The second challenge is met with a proposed Cuckoo Search-based offloading algorithm for IoT networks within an edge computing framework, designed to enhance service time and resource utilization for data-sensitive, deadline-driven tasks. The third challenge is resolved by offering a job scheduling algorithm that employs Cuckoo Search for IoT systems, concentrating on job priority, dependencies, and conflicts to create an effective job schedule.

All algorithms presented in this thesis have been validated through simulations and demonstrate superior performance compared to state-of-the-art techniques. The proposed edge server placement algorithm significantly surpasses existing IoT clustering protocols, even when more than 20% of nodes experience power loss. A benchmark dataset has been utilized for simulation purposes, along with effective simulation software including Matlab, I-Net, and Cisco Packet Tracer. Additionally, the proposed task-offloading method results in 12.8% fewer task cancellations compared to a state-of-the-art approach. Simulations indicate that the proposed task scheduling strategy outperforms existing edge-computing task scheduling methods, especially in scenarios involving conflicting or dependent jobs.

Nomeclature

g_i:i_{th} Glow worm

t:Unit of Time

$l_{g_i}(t)$:Luceiferin value of ith Glowworm at time t

ρ :luciferin decay constant

γ :luciferin enhancement constant

$r_{g_i}(t)$:Neighbourhood Range of i_{th} Glowworm at time t

r_s :Sensor range

r_d(t):Decision Range at time t

n_d :Desired number of neighbours of a sensor

$x_{g_i}.(t)$:Location of g_i at time t

$iter_{max}$:Maximum iteration number

s_i: i_{th} sensor node

NoS: Number of sensors deployed

$v_{s_i}(t)$:Voting index of i_th sensor at time t

$e_{s_i}(t)$:Energy Level of i_th sensor at time t

CH:Set of Cluster Head nodes

c_k:Cluster head of k_{th} cluster

n_k:Number of members of k_{th} cluster

M: Desired number of nodes in each cluster

w, x, y, z:Weighted constants for calculating luciferin value of a glowworm using Modified GSO algorithm

$s_i.|H_t|$: Number of neighbour nodes of s_i within its decision range at time t

$s_i.N(id, d)$:Set of all one hop neighbour id of s_i with distance d

IoT_{id}:Unique identity of each IoT node

$Edge_{id}$:Unique identity of each edge node $PoS(x, y)$:Position of the node

$Task_{id}$:Unique identity of each task

α:CPU Cycle Per Unit Time for IoT

β : Energy required by IoT device per unit time

γ : Bandwidth

δ : CPU Cycle per Unit Time for ECU

ϵ : required per unit time for transferring data

ζ : Energy required by ECU device per unit time

χ, ψ, ω:Weighted contants for Modified Cuckoo Search algorithm for task offlloading

VM_i : Number of virtual Machines in ith ECU

$jVM_{i,r}$: List of jobs placed in ith ECU w.r.t round r

L : Number of ECU

DUR_j : Execution time of jth job

RVM_j : Required number of VM for execution of the jth job

P_j : Priority sequence number of the j^{th} job.

S_j : State of the j^{th} job.

D_j :Dependency list of the jth job

C_j :Conflict list of the jth job

RD_j :Execution ready state of the job

μ : Power required for active ECU

ν: Power required for active VMs

Γ : Power required for inactive VMs in active ECUs

$AE_{i,r}$: Status of ith ECU w.r.t round r

$AVM_{i,j,r}$:Status of jth VM of ith ECU w.r.t round r

TE_r :Total energy consumed in round r

NE_r:Normalized energy consumed in round r

T :Total time for execution of all jobs

EC:Total energy consumed in round r

NRU_r : Normalized Resource utilization in round r

Φ :Constant multiplied with Energy Parameter

Ψ : Constant multiplied with Resource Utilization Parameter

$CJ_{i,j,r}$: List of compatible jobs of job j in ith ECU w.r.t round r

Acronym

AI: Artificial Intelligence

AP: Access Point

AR: Augmented Reality

CDN: Content Delivery Network

CH: Cluster Head

CISCO: Computer Information System Company

CNCF: Cloud Native Computing Foundation

CSA: Cuckoo Search Algorithm

CSJS: Cuckoo Search Based Job Scheduling

DTEN: Digital Twin Edge Network

ECU: Edge Computing Unit

ETSI: European Telecommunications Standards Institute

FCN: Fog Computing Node

FiCA: Firefly Clustering Approach

GA: Genetic Algorithm

GSO: Glowworm Swarm Optimization

HEED: Hybrid Energy Efficient Distributed Clustering

IoT: Internet of Things

IP: Internet Protocol

LEACH: Low Energy Adaptive Clustering Hierarchy protocol

MEC: Multi-access Edge Computing

OMNET: Objective Modular Network Testbed

QoS: Quality of Service

SI: Swarm Intelligence

TOCS: Task Offloading based on Cuckoo Search

VANET: Vehicular ad hoc network

WOA: Whale Optimisation Algorithm

WSN: Wireless Sensor Network

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Internet of Things (IoT) paradigm has enabled automated and convenient modern day lifestyle that has given rise to a plethora of data-intensive smart city applications [1]. The large scale proliferation of IoT devices span from smartphones, smart vehicles, drones to Industrial equipments, and beyond [3].

Intelligent monitoring systems that keep an eye on a variety of environmental, agricultural, health, and security parameters, as well as industrial control, are among the many useful applications of IoT that are gaining popularity [4].

An IoT device typically contains sensors that sense data and communicate to a remote server over the Internet. It is important to efficiently store, retrieve, and process the data sensed by the IoT devices. The conventional method involves sending the real life data gathered from multiple sensors, to cloud servers for additional processing and archiving. Sometimes, the data is locally processed, in limited manner before transferring it to a cloud server for storage and further processing. However, with wide variety of IoT applications in place, such an approach would incur huge Internet traffic and latency issues.

This highlights the necessity for a different paradigm that can carry out computations nearer to the sensors or data sources. Interestingly, edge computing enables the processing of IoT data closer to the sources, in a local network such as base stations. Such network edge infrastructure can aggregate data, process it with low latency, and then send it to the cloud for additional analysis if required. These nodes receive responses from the cloud's downstream processes as well. The infrastructure of edge computing not only minimises

latency in networking and computation but also maintains data privacy. Edge computing preserves the location awareness thus, location sensitive data processing could be conveniently executed on an edge server. The rise of IoT edge communication technology comes from addressing these issues. Such a system can be described using a three-tier architecture (as shown in Figure 1.1) where in IoT layer sends data to the local edge servers for processing. At the edge, it receives various job requests and data from the IoT layer. Then, the edge decides to forward them to the cloud server or process them locally and disseminates the results.

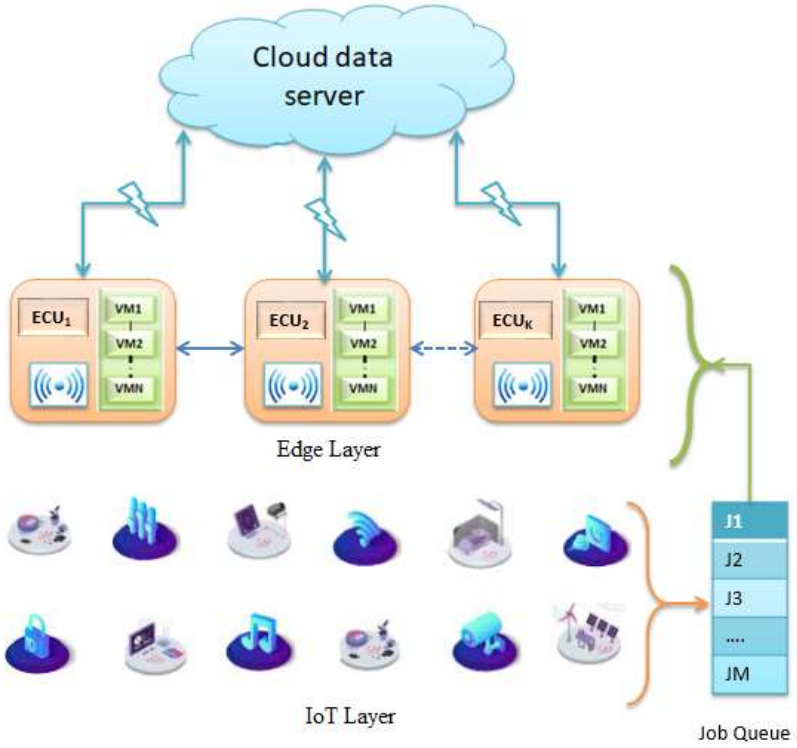


Figure 1.1: Three layer IoT-Edge network architecture

This thesis delves into the challenges and issues faced by such edge servers regarding IoT devices' task execution.

In this chapter we first summarize the evolution of the IoT-edge paradigm. Relevant applications are detailed in Section 1.2. Then the background of the thesis work is briefly discussed in Section 1.3 that leads to the motivation of the thesis discussed in Section 1.4. The main contributions of the work is then presented in Section 1.5. Finally, the organization of the thesis has been discussed.

1.1 Evolution of IoT-Edge Computing

The first evidence of the potential benefits of edge computing for mobile computing was presented in 1997, by Brian Noble and associates [5]. This marks the beginning of the evolution of edge computing. They demonstrated how delegating computation to a nearby server allowed speech recognition to be implemented with adequate efficiency on a mobile device with limited resources.

In order to deliver content closer to the end user, Akamai introduced nodes at strategic locations by launching its content delivery network (CDN)[6] in 1998. The strategy used by Brian Noble and colleagues was expanded in 1999 in an effort to increase battery life [7]. The practice of maximising a mobile device's computing power by utilising neighbouring infrastructure was first coined to as "cyber foraging" in 2001 [8].

In their 2002 paper, "Globally Distributed Content Delivery," [9] Akamai explained how networks can leverage edge delivery to provide content, aiming to address service outages and bottlenecks. In 2005, function caching was introduced [10] for managing personalized mailboxes to minimize latency. To reduce dependency on Cloud servers and make Cloud services more accessible to end users, Mahadev Satyanarayanan developed and introduced Cloudlet technology in 2009 [11].

Cisco coined the phrase "fog computing" first in 2012 [12]. The concept behind fog computing is to put a lightweight, cloud-like infrastructure close to mobile users. This allows the Fog to provide a direct, short-fat connection instead of a long-thin one for mobile users.

The Chinese Academy of Sciences introduced the idea of "Cloud-Sea Computing" in 2012 [13]. When used in this context, the term "Sea" describes an improved client side consisting of technologies and subsystems that communicate with both the human and physical worlds. Cloud-sea computing revolves around the concepts of "sea" and "cloud."

The Application Service Platform for Networks was co-developed in 2013 by IBM and Nokia Siemens Networks [14], which is when the term MEC initially appeared. This platform enabled mobile operators to install, run, and integrate applications at the network's edge.

The European Telecommunications Standards Institute (ETSI) created the MEC tech-

nical white paper in 2014 [15] and formed a new Industry Specification Group within ETSI to generate specifications. With the intention of expanding its application to heterogeneous access technologies (such as LTE, 5G, WiFi, and dixed access technologies), ETSI renamed the technology multi-access edge computing (still using the acronym MEC) in 2016 [16].

Another project launched in 2017 is the Linux EdgeX Foundry [17], a freely-accessible, platform-independent initiative managed by The Linux Foundation. The aim is to create an open, transparent, and shared edge computing framework inside the IoT. To meet the demands of the automotive big data industry, Automotive ECC (AECC)[18] was founded in January 2018.

2018 also saw the collaboration of the Eclipse Foundation and the Cloud Native Computing Foundation (CNCF) Foundation to introduce Kubernetes [19]. It is a popular cloud computing platform for ultralarge-scale environments, to the Internet of[Things edge computing landscape. In 2019, Edge AI - a brand-new technology was released [20]. With the help of this technology, AI can operate on edge devices like drones, robots, smartphones, and other IoT gadgets.

In 2020, the Digital Twin Edge Network (DITEN) is launched [21]. DITEN aims to combine digital twin (DT) technology with multi-access edge computing (MEC) to allow users of sixth generation networks to configure systems in real time and allocate resources flexibly.

1.2 IoT-Edge Computing applications

The applications of IoT-edge-cloud computing can be divided into four primary groups [22, 23]. As shown in Fig.1.2 the categories of IoT Edge applications are listed in the following order:

(a) Cloudlet based IoT Edge Applications

A possible way to characterise a Cloudlet is as an Internet-connected, reliable group of Computers that provide resources accessible to nearby mobile devices.Essentially, a Cloudlet is a form of virtual machine that functions like a miniature ”data center,”

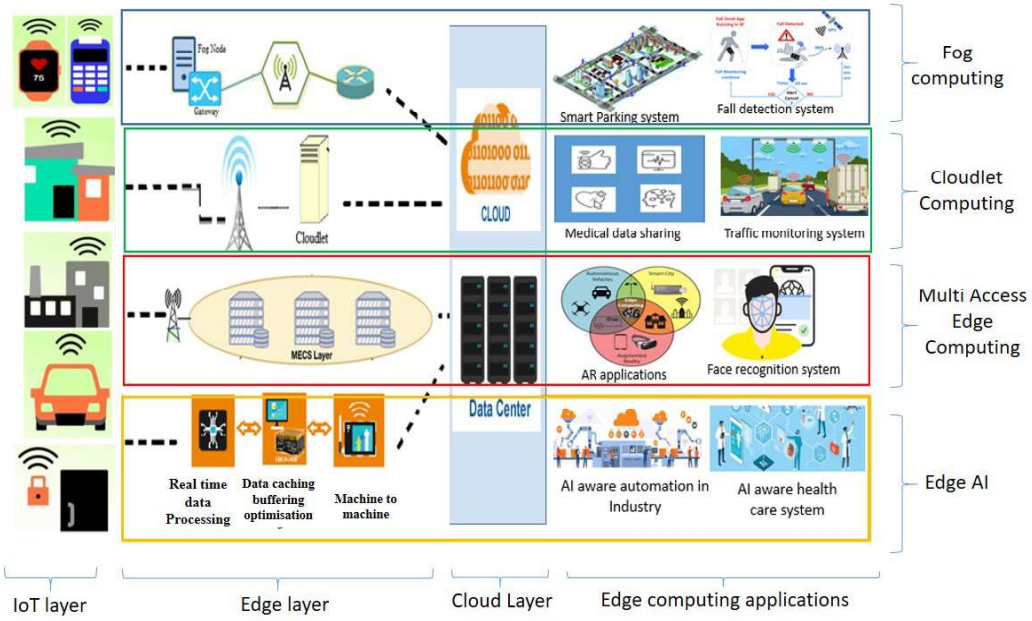


Figure 1.2: Categorisation of IoT-Edge computing applications

enabling users and devices to swiftly access resources through a WLAN [25]. Smart traffic monitoring system is an example application domain that utilizes cloudlets for observing traffic conditions through an ad-hoc network at some specific location as discussed in [24].

Fog computing in IoT edge applications involves a decentralized computing setup that utilizes Fog Computing Nodes (FCNs). These nodes can be positioned at multiple locations within the system, acting as an intermediary between the cloud and the endpoint devices [24]. FCNs are heterogeneous in nature. Set-top boxes, access points, routers, switches, and IoT gateways are just a few of the parts that can be used to construct them. Listed below are some instances of IoT-edge applications based on fog computing:

- A smartphone-based emergency alert system that preprocesses and offloads data from sensors to fog nodes for computation [26].
- A fall-detection algorithm and an implementation of it that makes use of a network of Fog nodes' computational power [27].
- A parking service that makes use of the distributed nature of Fog nodes by

combining data gathered from multiple Fog devices to locate the best parking space [28, 29].

(b) MEC based IoT Edge Applications

To reduce latency and improve context awareness, MEC is a form of Edge Computing that shifts processing and storage capabilities closer to the edge of the Radio Access Network. In general, the Radio Network Controller or a macro base-station are co-located with the MEC nodes, or servers [24]. Presented below are a few MEC-based IoT-Edge applications:

- Applications for augmented reality (AR), which merge computer-generated data with the real world [30]
- A face recognition application, in which the other components could be offloaded for cloud processing, but the image acquisition component must run on the mobile device to support the user interface [31] .

(c) Edge-AI based IoT Edge Applications

Edge AI refers to using artificial intelligence (AI) data closer to the point of source. To gather and handle the data locally, Edge AI transfers its computational resources [32, 35]. AI-driven localised data processing enables devices to make decisions instantly and effectively. A few Edge AI-based IoT-Edge applications are shown below:

- AI aware power allocation in industrial systems [33].
- Race car engine visualisation to identify required maintenance such as the component that is about to burn out[34].

1.3 Background

To provide the applications mentioned earlier, IoT devices located in a specific area can gather and send the collected or contextual data to servers via the Internet, either at regular intervals or in real-time. Data can vary in size from minimal quantities, like temperature or humidity measurements, to larger formats, including images [36]. Having every IoT device communicating with the server via a separate connection might lead to

an overbearing need for Internet connections and consumption of the network's wireless resources in an expansive IoT network.

One of the key components of the IoT is clustering [37]. The optimisation of communication between physical objects, preserving scarce resources, improving communication between nearby peers, and delivering better services all depend on an efficient topology. This is because, although the IoT appears to be an opportunistic network, not everything in the actual world is IP-enabled [38]. Because of its well-known benefits, clustering has been acknowledged by the scientific community as an effective strategy for achieving these goals. Clustering enhances energy-efficient routing by decreasing the number of nodes participating in the route formation process. It also allows data aggregation and provides Quality of Service(QoS) [38]. The literature also discusses clustering within a multi-layer IoT architecture, in which certain IP-enabled nodes in one layer connect to the cluster heads of sensing devices in the layer below [39]. Additionally, clustering maximises a network's lifespan for large-scale deployments and enhances network scalability. Ongoing research in IoT clustering e.g., [2][3][40][41] is leveraging advanced studies in Wireless Sensor Networks (WSNs), especially considering that sensors are fundamental elements of both IoT and WSNs. To realize the many advantages of clustering algorithms, cutting-edge technologies have been identified to adopt the cluster head selection method. Given that cluster heads are anticipated to possess significant processing capabilities and handle a greater communication load, various methods have been developed to optimize the selection of the cluster head [43][44][45]. Various bio-inspired algorithms designed for clustering are based on principles observed in living systems [41][46][47]. Particle Swarm Optimization (PSO), Arificial Bee Colony approach, Glow-worm swarm optimisation(GSO) technique have been used in many techniques for this purpose. Limited research has suggested firefly-inspired algorithms where "real world entities" engage in fierce competition to become Cluster Heads (CHs), while also seeking to draw in other participants to their clusters. Given that sensors play a crucial role in the expanding field of the IoT [48], these algorithms are anticipated to operate effectively in a dynamic environment while adapting to context shifts and maximizing the use of scarce resources [49].

The upcoming mobile networks designed for smart living require not only fast data transmission rates but also the necessary storage infrastructure for the associated applications ([50]). Nowadays, applications of mobile devices are becoming ever more demanding on computational resources. Hence, researchers found offloading to be an effective approach where some of the tasks can be assigned to the remote resource-rich cloud environment ([50][51]). However, this could introduce significant data transfer delays due to increased internet traffic between users and the cloud. Since the delay is undesirable, particularly for the mobile applications that require immediate response to the user, research has been redirected towards cloudlet-based offloading approaches as discussed by [52]. Cloudlets reside reasonably nearby the consumers employing Wi-Fi Access Points (APs) ([50][52]) though they are less resourceful than a cloud server. This drives the development of the field of mobile edge computing, as mentioned in [53]. With mobile edge computing, edge servers are placed close to where mobile devices are deployed i.e. the data source in an effort to lessen long latency and improve the present network architecture ([54]). The precise location of the edge server is a significant issue in the implementation of mobile edge computing architecture because choosing an acceptable location for the edge servers is vital and crucial [55, 56]. Different issues such as workload balance, access delay, user coverage, network robustness, capacity constraint, number of edge servers, and so on are taken into account for estimating optimal solutions. Research is primarily conducted with the goal of placing edge servers in order to enhance the experience for mobile users [50]. In addition, a few works also target finding a set of placement locations to optimize the server's performance. A few optimization techniques such as particle swarm optimization ([54]),genetic algorithm ([57]), cuckoo-search ([58]) based meta-heuristic approaches are mainly used in this regard. These approaches mainly take into consideration the problem of access delay. However, edge placement strategy while minimizing the total no of edge servers and hence, optimizing their coverage should be investigated.

By utilising low network latency, IoT task offloading to the edge rather than the cloud prevents high network congestion and shortens the response time for data analysis. In order to bridge latency gaps, edge computing offers a collection of remotely functioned computational and storage facilities at the edge of the network that are close to IoT de-

vices [59, 60]. More effective services for streaming, especially those that are bandwidth-intensive and prone to latency, can also be served by it. Additionally, the framework for edge computing minimises the overall service period by avoiding both the uploading and downloading of large files as well as the preparatory processing of tasks that are to be offloaded [61, 62].Moreover, a number of factors affect the recommendation of a workable model for task offloading on the IoT-edge system. The factors may include limited resources, prioritising tasks to be offloaded, evenly workload distribution among different edge servers etc [63, 64]. To improve overall service efficiency and reduce task offloading delays, it is necessary to propose an effective task offloading approach for managing IoT-edge computing resources and processing computation workloads for deadline-sensitive applications. Certain works, like the one reported in [65], focus on energy-efficient offloading, whereas the work reported in [66] takes deadline awareness into account.Few studies [67, 68] have been done on the issue of task offloading in edge computing environments based on IoT considering deadline aware tasks using meta-heuristics algorithms.

The diverse data standards from various IoT devices give rise to specific service requirements, making the placement of IoT services in edge computing critical [69, 70, 71]. Given the geographically dispersed nature of IoT services across the deployment area, edge computing units (ECUs) may experience either underload or overload situations that can impact service performance [72, 73]. Additionally, with the widespread deployment of ECUs, optimizing energy costs is essential for the sustainable growth of the IoT industry. Moreover, certain IoT services might conflict with one another, necessitating that they not be processed on the same ECU for security reasons [74, 75].

Conversely, certain IoT services may encounter conflicts, which could prevent them from being processed within the same ECU due to security concerns. In addition to conflicts, there can also be dependencies between IoT services [76]. Therefore, it is crucial to maintain the execution order at the edge, allowing the output of one task to serve as the input for its dependent task. To ensure QoS for IoT services across ECUs, it is essential to address privacy conflicts and uphold job dependencies during the placement of IoT services [77, 78, 79]. Priority is another aspect that should be looked into in order to address the different QoS requirements of the various IoT services. Specifically, real-time services

must be given top priority to ensure QoS guarantees. Despite these insights, optimizing the overall execution performance of Electronic Control Units (ECUs) concerning resource usage and power consumption for IoT service placement remains a challenge [80, 81, 82]. There is a limited amount of research on job scheduling specifically within IoT-based edge computing environments. Some works are based on job priority as in [83] while the work reported in [74] consider resource utilization.

From this background, we could find that research works are going on for the IoT-Edge-Cloud paradigm and various aspects of it has been looked into. This sets the importance of the field and emphasizes the need for further research into the practicality of the domain. This leads to the motivation of the thesis discussed in the next section.

1.4 Motivation

Resource management at the edge for enhancing performance at the IoT devices is a key research challenge. This gives rise to three crucial research issues- (i) edge server placement, (ii) IoT task offloading strategy at the edge or cloud, (iii) task scheduling at the edge servers. The main performance parameters subject to which the strategies should be evaluated are (i) energy efficiency, and (ii) resource utilization. Also, to formulate an effective strategy, it is important to consider the nature of the tasks such as, their dependency or conflict, deadline awareness of the tasks that are mostly not considered by the state-of-the art works. Interestingly, behaviour of such influencing parameters are often contradictory such as lowering energy consumption of the devices may result in delay in execution.

The need for resource optimization in these contexts is highlighted by several factors, including the rising demand for low-latency applications, the need for scalability in dynamic settings, and the necessity to manage network congestion. As IoT ecosystems continue to develop, optimizing resource utilization becomes not just a technical necessity but a strategic priority for delivering responsive and sustainable smart applications.

The above factors put the focus on the following research challenge.
"What should be the optimal strategy for executing the tasks generated from the IoT nodes in IoT-edge environment subject to various QoS parameters ?"

Consequently, the above research challenge is mapped to the below research questions.

- What is the best approach for placing edge server nodes in a specific area to ensure that the entire region is efficiently covered by an ideal number of non-overlapping clusters, regardless of the density of IoT nodes present?
- What is the best way to offload deadline aware data centric tasks from IoT devices to edge servers while ensuring minimal job cancellations?
- What is the optimal way to arrange job schedules considering different levels of service qualities such as job priority, dependencies, and conflicts among jobs?

1.5 Contribution

Inspired by the research challenges and motivation stated above, in this thesis resource optimised task execution strategies in IoT-Edge environment have been designed.In view of the above challenges, the candidate solution space becomes exponentially large while considering the influencing factors that are often contradictory. This motivates the usage of meta-heuristic optimization algorithms as they enable exploration and exploitation across the search space. The main contributions are stated as follows:

- Cluster head selection and Edge server placement algorithm

  A clustering algorithm utilizing modified GSO (Glow-worm Swarm Optimization) is proposed, capable of partitioning the IoT network into an optimized number of clusters to minimize communication overhead and enhance the network's longevity. The suggested approach involves repeatedly selecting cluster heads to ensure an even distribution of load across the nodes. Additionally, the geographical placement of the cluster heads significantly impacts the total energy consumption of the network.

  Additionally, the challenge of strategically positioning edge servers to optimize workload distribution while enhancing the coverage of base stations is framed as a constrained optimization problem. We have introduced a modified GSO algorithm to address this issue.

- Optimal task offloading algorithm ensuring minimal task cancellation for IoT-Edge Environment

  After strategically positioning the edge server nodes and grouping the area of interest into distinct clusters, the primary concern is finding the best task offloading approach.

  Therefore, this thesis pays attention to develop an optimal task offloading strategy. A Cuckoo search optimization based task offloading algorithm (TOCS) is proposed for IoT edge environment that minimizes task failure rate of deadline aware tasks. A task can either be performed directly on the device or transferred to an ECU located in the same or a different cluster. The offloading decision depends on the following three factors

  - Deadline
  - Necessary data required for process finalization, gathered and transmitted from end device
  - CPU cycle needed for execution

- Job Scheduling approach ensuring maximum resource utilization and minimised energy consumption for IoT-Edge Environment. After the tasks are effectively delegated to the edge servers, it is clear that a task scheduling strategy is essential to ensure that the tasks are finished successfully within the designated timeframe. This approach should aim to create an optimal schedule for tasks with diverse QoS needs.

  Accordingly, a task scheduling approach is proposed applying cuckoo search optimization technique considering the following factors

  - Job priority,
  - Job conflicts, and
  - Heterogeneous ECU capacities.

  The proposed algorithm produces a job schedule in a way to optimize the overall ECU performance in terms of both resource utilization and energy efficiency.

In summary, the key contributions made by the thesis are as follows:

- Creation of a cost-effective algorithm for selecting cluster heads and placing edge servers to enhance network longevity.
- Development of an optimal task offloading algorithm that minimizes task cancellations in the IoT edge environment.
- Design of a job scheduling method that maximizes resource utilization while reducing energy consumption in the IoT-Edge environment.

1.6 Organization of the Thesis

The organization of the thesis is as follows:

Chapter 2 provides a review of the essential background information needed to comprehend the problem statement and potential solutions discussed in subsequent chapters. An overview is presented on how existing studies approach these issues, along with an in-depth examination of various execution strategies documented in research. The primary emphasis is on tackling the issue of creating effective and energy-efficient strategies for executing tasks in the IoT-Edge environment.

In Chapter 3, a cluster head selection algorithm relying on Glow worm swarm optimization is introduced. The algorithm efficiently categorizes the network into an optimal number of clusters while ensuring a balance between the edge server's operational needs and the base stations covered by an edge. The issue of edge server placement is also investigated.

Chapter 4 introduces a task offloading algorithm utilizing the Cuckoo Search Algorithm within an edge computing framework for the IoTs. This algorithm is designed to optimize service time and resource utilization for applications with data-centric, deadline-oriented tasks. Additionally, it aims to reduce energy consumption and the occurrence of task failures in the IoT-Edge environment.

In Chapter 5 a Cuckoo Search-based job scheduling algorithm is presented, designed for IoTs in an edge computing environment. This algorithm is able to generate an efficient

job schedule considering factors like job priority, dependencies, and conflicts. It aims to optimize resource usage and energy consumption.

Chapter 6 summarizes the contributions of this thesis and presents the concluding remarks. The scope for future research is also identified.

Chapter 2

Literature Survey

This chapter reviews the essential background that incites to realize the problem formulation and the solutions as presented in subsequent chapters. A study is made to understand the challenges associated with IoT-Edge network task execution. A brief survey is carried out about how the existing studies address these challenges. Besides, few execution strategies found in the literature are also studied in detail. Focusing on these aspects is essential to address the problem of designing energy efficient optimal task execution strategies for IoT-Edge environment.

The works are grouped along two main research directions related to the thesis- (i) grouping of the IoT nodes into clusters to be covered by an edge and (ii) task offloading and scheduling strategies at the edge and cloud servers.

2.1 Survey on IoT clustering techniques

This section provides a brief overview of the current clustering protocols relevant to IoT and WSN. Several works are proposed in the literature to enable the advantageous features of clustering in the above mentioned networks.

In MHCM[40], the authors introduce a clustering algorithm aimed at reducing internet connections while achieving optimal delay. This method focuses on choosing the smallest group of coordinators that can effectively cover all IoT nodes in a network and optimizing the overall number of coordinators selected.

A Balanced Energy-Efficient clustering algorithm, BEE,is proposed in SBEEM[2],

which identifies cluster heads by considering energy consumption and sensor layout. This approach improves the network's longevity and maintains reliable coverage. Additionally, the algorithm employs MIMO for data transmission and utilizes a single communication interface for the sensing nodes.

Swarm intelligence algorithms, like ASFiCA [41], have been suggested for choosing cluster heads in IoT systems. These algorithms, which are inspired by firefly behavior, facilitate self-adaptation and the selection of devices based on their performance in the network and the particular deployment region.

In EPMS [84], the authors present a clustering algorithm for wireless sensor networks that utilizes particle swarm optimization (PSO). This method segments the network into a predetermined number of clusters, with the PSO algorithm helping to establish the boundaries of these regions. The selection of the cluster head is based on its closeness to the center of gravity and the overall distribution of energy.

UCRA-GSO [85] is a clustering routing algorithm based on GSO that takes into account various factors such as local density, distance, energy usage, and the distribution of cluster heads. It selects cluster heads with greater residual energy and employs the GSO algorithm to determine the optimal clustering strategy. When the communication range is shorter than the distance, the next hop is selected accordingly.

Bozorgi's research [86] presents a technique utilizing the Whale Optimisation Algorithm (WOA) aimed at prolonging the system's lifespan. Cluster heads positioned close to the base station possess greater energy for transmitting data. The clustering phase decreases the number of messages exchanged during metarounds, employing a multi-hop method for communication between cluster heads and the base station.

The Chicken Swarm Optimisation based Clustering Algorithm (CSOCA) is suggested in [87] to raise WSNs' energy efficiency. By using a sigmoid function on each individual, the optimisation of the chicken swarm is discretized. Crossover and mutation processes are used by CSOCA-GA by incorporating the Genetic Algorithm's procedures into CSOCA.

Most existing clustering algorithms focus on maintaining the number of sensors that are still alive to extend the longevity, ignoring the distribution of the sensors [88][89][90]. Again it is observed that most of the existing approaches groups nodes into clusters containing a

fixed number of nodes. This does not apply in some situations, particularly when a large number of nodes are inactive. The coverage of a network is largely influenced by how sensors are distributed, which is essential in various systems, including smart cities, smart agriculture, environmental monitoring, and healthcare [91][92][93][94].

Table 2.1 provides a summary of the main characteristics of the previously mentioned leading clustering techniques, organized by publication year, methods used, energy Consumed, latency, coverage, network lifetime taken into account. It is evident that most of the existing clustering algorithms focus on maintaining the number of sensors that are still alive to extend the longevity, overlooking the distribution of these sensors. However, the coverage and lifespan of a network is highly influenced by the sensor distribution, and it is crucial in most systems, like in smart city, smart agriculture, smart environment monitoring, healthcare systems, etc.

Table 2.1: Comparison between existing clustering algorithms

Algorithm	Energy	Latency	Coverage	Network Lifetime
EPMS,2017[84]	✓	✗	✗	✗
ASFiCA,2017[41]	✓	✓	✗	✗
MHCM,2018[40]	✓	✗	✗	✗
SBEEM, 2018[2]	✓	✗	✓	✗
UCRA-GSO,2019[85]	✓	✗	✗	✗
CSOCA-GA ,2020[87]	✓	✗	✗	✓
WOA,2021[86]	✓	✗	✓	✓

The capacity to collect, store, and process vast amount of data has increased with the development of IoT devices. Scalability becomes imperative in light of this. More processing tasks may be moved closer to the locations where the data is collected by Internet of Things(IoT) devices by using edge computing. The edge computing units (ECUs) may be used for processing tasks in this situation. The location of the edge servers optimally presents the most significant challenge in the implementation of mobile edge computing architecture since finding an appropriate location for the servers is fundamental and critical.

The authors of [95] present a formulation of mixed integer linear programming that aims to minimise the cost of deploying edge devices by simultaneously meeting a target level of computational demand and network coverage.

In order to balance the workloads of edge servers and reduce access latency between the mobile user and edge server, the authors of [50] developed the edge server placement problem as a multi-objective constraint optimisation problem. Edge servers are then placed in certain strategic locations. The most optimal solution is subsequently identified using mixed integer programming.

In [96], authors have considered two issues while deploying of edge servers. The first concern is the cost of deployment, and the second is the coverage area of the edge servers. To address the first concern, a dynamic programming algorithm is suggested to find a solution. For the second concern, a geometric image method is utilized to determine the area covered by the edge server.

The objective of the edge server placement issue, as outlined by the authors in [97], is to reduce the communication delay for mobile users while also distributing the workload evenly across edge clouds. The authors have suggested an approximate method that uses mixed-integer quadratic programming and K-means to solve the issue.

In [98], the authors concentrate on enhancing the deployment of diverse edge servers to improve network response time. They employ an offline strategy and a mobility-aware game-theory approach to account for user movement.

In [99], the authors integrated a greedy algorithm with a genetic algorithm (GA) to reduce the number of edge servers while ensuring load balancing among them and satisfying the quality of service (QoS) needs of mobile users.

In [100], the authors identified the ideal locations for edge servers as clustering centers using an enhanced Glowworm Swarm Optimization (GSO) algorithm. Each base station in the neighboring list of an edge server is allocated to that server. The placement strategy is achieved by minimizing the distance users need to travel to access their edge server and by effectively distributing the workload.

In [101], authors have proposed a preference-aware edge server placement strategy that provides more equitable workload distribution by minimising query latency and evenly distributing the load among edge servers. For large-scale data sets, the authors have proposed a heuristic algorithm called TAKG (TAbu search with K -means and Genetic algorithm) to address the edge server placement problem.

The authors of [102] have identified three crucial challenges in deploying edge servers: edge location, user association, and capacity. They introduced an optimization algorithm for edge server deployment that emphasizes user distribution density, identifies optimal locations, and ensures the installation of required nodes, all aimed at minimizing costs and construction expenses.

The authors in [103] developed a profit model incorporating energy consumption and access delay for edge servers, utilizing the 5G user plane function (UPF) and particle swarm optimization to maximize profits.

The Table 2.2 presents an overview of the key features of above stated leading-edge edge server placement techniques categorised by publication year, techniques, and goals.

Table 2.2: Comparison with State of the art Edge Placement approaches

Paper	Year of Publication	Technique Used	Load Balancing	Network Coverage	Energy	Number of Servers
[95]	2018	Mixed Integer Programming	✗	✗	✗	✗
[50]	2019	Mixed Integer Programming	✓	✓	✗	✗
[96]	2019	Dynamic programming	✗	✗	✗	✓
[97]	2020	Mixed Integer Programming, Kmeans	✓	✓	✗	✗
[98]	2020	Integer Programming, Game theory	✗	✗	✗	✗
[99]	2021	Integer Programming, Genetic algorithm	✓	✗	✗	✓
[100]	2021	Glowworm Swarm Optimization	✓	✗	✗	✗
[101]	2022	Integer Programming, Genetic algorithm, TABU	✓	✓	✗	✗
[102]	2022	Mixed Integer Programming	✗	✗	✗	✗
[103]	2023	5G user plane function (UPF), Particle Swarm Optimization	✗	✓	✓	✗

Recent studies primarily aim at improving specific metrics such as load balancing and energy consumption, but lacks holistic approaches that address the interdependencies among these metrics. Balancing multiple factors like load, energy, and coverage simultaneously is challenging.

2.2 Survey on task execution strategies for IoT-Edge-Cloud Architecture

Owing to the requirement for job response, an effective task offloading plan is necessary. The prevailing task offloading protocols that are relevant in the IoT environment are addressed briefly in this section. The works also emphasise how important it is to conduct edge computing research regarding IoT task offloading because of its distinct advantages over cloud computing.

In order to jointly optimize the energy consumption and the offloading probability, the authors in [104] takes multiple offloading into account through collaboration between fog nodes.The strategy hasn't, however, taken into account computational cost optimization or data-centric IoT applications.

The authors in [105] present a blockchain-based architecture for IoT-Edge-Cloud computing that employs an energy-efficient dynamic task offloading algorithm and Lyapunov optimization methods. Their focus is on reducing energy usage and task response times, although they do not take into account data-centric applications or the potential for task deadline failures.

In [106], the authors proposed a task offloading strategy based on fuzzy logic for latency-sensitive IoT applications in Edge-Cloud environments. This approach seeks to enhance resource utilization and reduce service time, but it does not address cost optimization.

The work presented in [67] introduces a clustered multi-UAV system designed for sharing and offloading IoT devices. It employs a multi-agent deep reinforcement learning method to satisfy quality of service (QoS) needs and lower the expenses associated with network analysis. Nonetheless, this method does not prioritize the reduction of task cancellations or address data-centric applications.

In [68], the authors introduce an offloading algorithm for IoTs that leverages a combination of meta reinforcement learning and deep reinforcement learning to improve computational efficiency and mobility; however, it does not address data-centric applications or the reduction of task deadline failures.

In [107], the authors present a framework for virtualizing IoT platforms that focuses on minimizing latency and efficiently managing edge nodes. However, it overlooks the aspects of energy and cost optimization.

In [108], authors have proposed a getaway-centric IoTs system to allow IoT devices to operate intelligently and autonomously in edge computing. Nevertheless, this approach does not take into account the equitable distribution of workload and deadlines for IoT services.

An intelligent Computational Offloading scheme for Dependent IoT Application (CODIA) is proposed in [109], which decouples the performance enhancement problem into two processes: scheduling and offloading. CODIA utilizes an Actor-Critic approach, enabling IoT devices to implement intelligent models and adapt their offloading strategies dynamically to minimize latency while managing energy usage. Nevertheless, this method does not take CPU-intensive data-centric tasks into account.

An algorithm for task offloading and scheduling based on the osmotic approach is proposed by the authors in [66]. The tasks and devices are categorised in the osmotic approach, and the tasks are then allocated to the best devices according to their dynamically available capacity. Nevertheless, this method does not take CPU-intensive data-centric tasks into account.

In [110], authors have presented a task offloading strategy that focuses on energy efficiency and meeting deadlines for mobile cloud workflows. It involves a task offloading decision model that considers This method employs a novel algorithm known as the adaptive inertia weight-based particle swarm optimization (NAIWPSO) to develop a channel constraint-based strategy (CC-NAIWPSO) that produces an energy-efficient offloading plan while adhering to deadlines. This method does not, however, take into account evenly distributing work, or reducing task deadline failure.

In [111], authors have proposed a multi-objective strategy for MEC offloading that satisfies users' various needs while utilising the biogeography-based optimisation (BBO) algorithm (the execution time, energy consumption and cost). This method does not, however, take into account data-centric IoTs applications, evenly distributing work, or reducing task deadline failure.

In order to jointly optimise task dependencies with deadline constraints for tasks that are delay-sensitive, [112] proposes a directed cyclic graph model that illustrates the dependencies among these tasks. Priority-aware scheduling is used to schedule dependent tasks while taking their deadlines into account.This method does not, however, take into account data-centric IoTs applications, evenly distributing work, or reducing task deadline failure.

Task offloading is modelled as a multi-objective optimization problem in [113], minimizing total system power consumption and end-to-end delay. Authors solve this problem using non-dominated sorting genetic algorithm (NSGA-II) and Bees algorithm due to its NP-hardness.This method does not, however, take into account data-centric IoTs applications, evenly distributing work, or reducing task deadline failure.

In [114], the study focuses on identifying the most suitable edge nodes for offloading 5G data, ensuring a balance in energy usage within advanced networks. The approach utilizes mobile edge computing, macro base stations, and small base stations to achieve energy-efficient offloading, employing the particle swarm optimization (PSO) algorithm for selecting the edge network.

Sophisticated stochastic optimization techniques are applied by the authors of [115] to convert the resource allocation and energy-efficient task offloading problem into a deterministic one. Nevertheless, deadline-conscious, data-centric, or CPU-demanding tasks are not taken into account by this approach.

The comparison among the presented works is displayed in Table 2.3. It is evident that previous research has used a variety of methods to optimise task offloading in IoT edge environments. Conditions like energy, cost, delay, and resource use have been prioritised. When defining the optimization approach, the majority of the works primarily took one or two issues into account. A few recent works take into consideration the tasks' deadline requirements, heterogeneous ECU environments, and datacentric tasks. However, the majority of the literature frequently disregarded any connections between these problems. These are crucial requirements, though, in order to offload tasks to the edge nodes. From examining the latest techniques, we can determine that many of the studies appear to focus on several key issues. However, there are no published works that consider the

Table 2.3: Comparison with State of the art task offloading technologies

Research Paper			Task Metrics			Optimization Metrics					Heteroginity	
Paper	Year of publication	Methodology used	data-centric	Deadline aware	CPU-demand	Energy	cost	Delay	task-cancellation	Work load distribution	task	Deployment
[104]	2020	successive convex approximation (SCA) and Dinkelbach method	✗	✗	✗	✓	✓	✓	✓	✗	✗	✗
[105]	2020	Lyapunov optimization technique	✗	✗	✗	✓	✓	✓	✗	✓	✗	✓
[67]	2021	multi-agent deep reinforcement learning (MADRL)	✗	✗	✗	✓	✓	✓	✗	✓	✗	✗
[68]	2021	metareinforcement learning (meta-RL) model	✗	✗	✗	✓	✓	✓	✗	✗	✗	✓
[108]	2022	deep learning	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓
[115]	2022	dynamic energy efficient task offloading and resource allocation (NTORA) algorithm	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗
[114]	2022	Particle Swarm Optimization (PSO)	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗
[66]	2023	osmotic approach	✗	✓	✗	✗	✗	✓	✗	✓	✗	✗
[107]	2023	grouped crossover genetic algorithm (GCGA)	✓	✗	✓	✗	✗	✓	✓	✓	✗	✗

following factors at the same time.

- Minimising deadline aware , data centric job cancellation
- Optimizing cost and resource utilization

Consequently, research on the aforementioned topic is required.

Implementing IoT services within Edge Computing Units (ECUs) effectively reduces the bandwidth load in IoT systems while also enhancing the quality of service (QoS) [116, 117, 118, 119]. Nonetheless, since the service requirements come from a range of IoT devices that have different data standards, placing IoT services and scheduling tasks in an edge computing environment is rather challenging. [69, 70, 71].

In this section, the existing job scheduling protocols applicable in the field of IoT are discussed briefly. The works also highlight the need for edge computing research w.r.t IoT job scheduling as its characteristics differ from that of cloud.

In [74], authors utilized the edge computing as a novel paradigm to process the IoT service placement in smart cities. They evaluated factors like load balance, energy consumption, and resource utilization to enhance the quality of IoT services in smart cities. To achieve balanced service placement strategies that optimize various performance met-

rics while preventing privacy breaches and meeting time constraints, they employed the Strength Pareto Evolutionary Algorithm (SPEA2) technique. The authors did not consider the dependency and priority constraint among jobs. Heterogeneous ECU is not considered in the work. In the study the job placement is given, job scheduling is not considered.

In a simulation study, the authors of [120] suggest a conceptual framework for fog resource provisioning. They presented the idea of a "fog cell," which is a software element operating on fog devices that manages and oversees a specific set of IoT devices. By utilizing this concept along with others, they create a model for orchestrating IoT devices through a layered cloud/fog resource management system and offer an appropriate resource allocation solution for task distribution among these devices. However, energy consumption was not considered in the study.

The research presented in [75] introduces a communication approach that minimizes overhead by informing fog nodes about adjacent replica nodes, allowing these replicas to manage requests rather than relying on cloud storage. Nonetheless, the study did not address energy consumption.

In [77], a Simulated Annealing Algorithm (SAA) is used to compare the performance of the GA in its centralized and distributed forms. SAA is another AI-based algorithm which can be used to solve NP complete optimization problems in a heuristic approach. They haven't considered any dependency and conflict constraint among the jobs.

In [78], the authors presented FOGPLAN, a framework designed for QoS-aware Dynamic Fog Service Provisioning (QDFSP). The framework focuses on efficiently managing application services on fog nodes to meet low latency and quality of service (QoS) requirements while minimizing costs.

In [80], the authors created an online service placement framework that takes mobility into account to optimize both latency and migration costs. They employed Lyapunov optimization to handle future system information and devised effective heuristic methods. However, this framework did not factor in energy consumption and resource utilization.

In their work [81], the authors introduced two strategies for orchestrating services in distributed edge systems: a flat architecture and a hierarchical recursive method. They

recommended employing OpenStack or incorporating an extra orchestration layer to optimize the allocation of service deployment requests. Nonetheless, this study concentrated exclusively on service placement, neglecting the processes of activating and deactivating servers.

In [121], the authors introduced a framework for resource management and task scheduling based on mobile edge computing (MEC-RMTS) to create an efficient data delivery mechanism that facilitates task offloading with low latency and scalability for smartphones connected to IoT networks. The framework employs power usage (PU) to reduce energy consumption while utilizing a convex gradient-dependent game model (GM) for data acquisition. Additionally, a mixed-intelligent, non-linear programming model is established to address common needs and resource utilization (CNRU) in order to minimize request delays at the endpoint devices. However, authors haven't considered any dependency and conflict constraint among the jobs.

The authors in [122] proposed a PSO-based heuristic strategy to solve the joint problem of service placement and task provisioning. This algorithm solved the resource scheduling problem by greedy-based and genetic-based algorithms. However, scarcity of resources in fog nodes, due to the main functions load, significantly degrades the platform's performances.

Table 2.4: Comparison with State of the art task scheduling technologies

Work and Year of Publication	Technique Used	Resource Allocation	Energy Conservation	Job Dependency	Job Conflict	Priority
[120] in 2017	GA	✓	✗	✗	✗	✗
[75] in 2018	RPA	✓	✗	✗	✗	✗
[78] in 2019	Greedy Algorithm based FOGPLAN	✗	✓	✗	✗	✗
[122] in 2019	PSO-based heuristic strategy	✓	✗	✗	✗	✗
[74] in 2020	SPEA2	✓	✓	✗	✓	✗
[123] in 2020	Clustering algorithm using betweenness centrality and k-means	✗	✓	✗	✗	✗
[121] in 2021	Gradient-dependent game model	✗	✓	✗	✗	✗
[124] in 2021	Reinforcement learning model	✗	✓	✗	✗	✗
[125] in 2022	Priority-aware Semi-Greedy algorithm (PSG)	✗	✓	✗	✗	✓
[126] in 2023	Dual-phase metaheuristic algorithm	✓	✓	✗	✗	✗
[127] in 2023	Electric earthworm optimization algorithm (EEOA)	✗	✓	✗	✗	✗

Table 2.4 presents a comparison with current state-of-the-art technologies. It is evident that the existing studies utilize meta-heuristic techniques to enhance edge performance, placing significance on factors such as quality of service (QoS), migration costs, and resource utilization. However, job priority, conflict and the dependency between the jobs remain mostly unexplored. Though, these are important criteria for scheduling jobs at the edge nodes.

2.3 Summary

In the IoT Edge job execution paradigm, three distinct areas have been thoroughly studied: edge server placement, job offloading, and task scheduling strategy. The potential impact of these factors on overall job execution quality in an IoT-Edge environment has been reviewed. While exploring state-of-the-art techniques related to these parameters, research gaps have been identified that require further exploration. No clustering mechanisms have been found that can determine the optimal number of clusters while ensuring network

lifetime prolongation and maximum coverage guarantee. Additionally, there is a need for edge server placement algorithms that can balance workload with the least number of server nodes. When optimizing resource usage, most task-offloading techniques do not consider minimizing task cancellation for deadline-aware tasks. Most job scheduling techniques prioritize factors such as QoS, migration cost, and resource utilization while neglecting job priority, conflicts, and dependencies between jobs. However, these factors are essential for the effective execution of jobs at the edge nodes. This inspired us to create the suggested job execution strategies for the IoT edge environment that are covered in the subsequent chapters.

Chapter 3

Edge Server Placement

3.1 Introduction

The chapter uses clustering to provide energy efficient routing by minimizing the number of participating nodes in the route formation. Moreover, proposed clustering approach also improves the network scalability and optimize lifetime of a network for large scale deployments. In the proposed architecture, IoT network devices are organized into clusters, with a designated representative known as the Cluster Head (CH) for each cluster. The CH gathers data from the other devices within its cluster and relays this information to the server, serving as a point of communication for the entire cluster rather than requiring individual Internet connections for each device. Conservation of the Internet connectivity and the wireless network resources could be made possible by the proposed architecture.

In this instance, edge servers are positioned near mobile devices to promote low latency and make the current network framework more proficient. The edge network gains computing and storage resources from the core network to provide quicker and more dependable service. However, the location of the edge servers optimally presents the most significant challenge in the implementation of mobile edge computing architecture since identifying a suitable location for the servers is essential and highly important. The cluster head positions obtained in the proposed methodology may be used for positioning edge servers. The main contributions of this chapter are:

- The chapter presents a clustering algorithm using modified GSO(Glow-worm swarm

optimization) which is able to divide the IoT network into optimised number of clusters, such that the communication overhead is minimum and the lifetime of the network is optimised.

- The proposed methodology repeats the cluster-head selection, so that, the load is uniformly distributed among the nodes. Again, the geographical distribution of the CHs severely influences the overall energy consumption of the network. For prolonging the lifetime of the network, the proposed algorithm ensures that the cluster heads are spread evenly.

- The issue of placing edge servers strategically to balance their workloads yet increase the coverage of base stations is proposed as a constraint optimization problem.

The rest of the chapter is organized as follows: 3.2 presents a brief description of the proposed methodology. Section 3.3 summarizes the experimental setup and analysis of the experimental results while Section 3.4 summarizes the chapter.

3.2 Proposed Methodology

The concept of GSO is applied here to solve the dynamic cluster formation and cluster head selection problem of IoT. The GSO mechanism is modified to suit the problem. Each IoT node is considered to be a glow worm, that is, a candidate solution. Our proposed methodology takes into account that some nodes may be dead, after certain round, therefore, rearranges the clusters to extend the lifetime of the network. Moreover, it ensures that the cluster heads are equally distributed in the region of interest. The proposed GSO-based cluster-head selection scheme is divided into three phases: sensor luciferin update phase, cluster formation phase, and neighbourhood range update phase. The overview of GSO is presented first. The proposed algorithm and its complexity analysis are presented in subsequent subsections.

3.2.1 Overview of Glow worm Swarm Optimization (GSO) algorithm

The GSO algorithm was first introduced in [128]. The agents in the GSO algorithm, called glow-worms carry a luminescence quantity called luciferin. Each glowworm is attracted

by the brighter glow of other neighboring glowworms. Based on this local decision ranges, eventually a global solution is reached. In GSO, a swarm consists of N agents known as glowworms. A state of a glowworm g_i at time t can be described by the following set of variables: a position in the search space $(x^{g_i}(t))$, a luciferin level $(l^{g_i}(t))$ and a neighbourhood range $(r^{g_i}(t))$. The GSO algorithm outlines the progression of these variables over time.

Initially, agents are randomly distributed in the search space. Other parameters are initialized by predefined constants. Each subsequent iteration consists of three phases: updating luciferin levels, moving the glowworms, and refreshing the neighborhood range.

To encode the fitness of the current position of a glowworm g_i in the luciferin level, the following formula is used:

$$l^{g_i}(t) = (1-\rho)l^{g_i}(t-1) + \gamma^J((x^{g_i}(t)) \tag{3.1}$$

where: ρ is the luciferin decay constant, γ is the luciferin enhancement constant and J is an objective function.

Then, each glowworm tries to find its neighbours. In GSO, a glow-worm g_j is a neighbour of a glow-worm g_i only if the distance between glow-worms g_i and g_j is shorter than the neighbourhood range $(r_{g_i}(t))$ and additionally, glowworm g_j has to shine brighter than $g_i(l_{g_j}(t) > l_{g_i}(t))$. If a glowworm has several neighbors, it selects one randomly, with the likelihood of selection being proportional to that neighbor's luciferin level. Eventually, the glowworm takes a step towards the chosen neighbor. Step size is constant and equals s.

In the last phase, the neighbourhood range $r_{g_i}(t)$ is updated in order to limit the range of the communication from an ensemble of agents. The following formula is used:

$$r_{g_i}(t+1) = min\left\{r_s, max\left[0, r_{g_i}(t) + \beta\left(n_d - |n_{g_i}(t)|\right)\right]\right\} \tag{3.2}$$

where: r_s is the sensor range (a constant, which limits the size of the neighbourhood range), n_d is the desired number of neighbours, $|n_{g_i}(t)|$ is the number of neighbours of a glowworm g_i at time t, and β is a model constant.

3.2.2 System model

Given a set of sensors, $S := \{s_1, s_2, ...s_{NoS}\}$, with the following properties.

- Sensing range of each node: r_s.
- Decision range of each node: r_d. Initially r_d=r_s.
- Initial voting index of each sensor : $v_0 = 0$.
- Initial energy level of each sensor = e_0.
- Initial Luciferin intensity of each sensor: l_0.
- Let CH:=$\{c_1, c_2, ...c_k\}$ denote the set of cluster heads and N:=$\{n_1, n_2, ...n_k\}$ denote the number of members in each cluster. Initially, each node is treated as cluster head, i.e, every node is forming a cluster with only one member, that is, n_i=1 $\forall$ i.
- Desired number of nodes in each cluster :=M, where M is dependent on the sensing radius of the deployed nodes.
- All the nodes have their unique id.

As a response of "Hello" message for neighbour discovery, nodes would send their unique id. The sender node is able to compute the distance depending upon the response time taken by the receiver node. There is no need to know the position of all the nodes.

After deployment of the nodes, the sink node gathers the initial properties of each node. After that, it initiates the cluster head selection algorithm for the first time. The selected cluster head information are sent to the nodes after successful execution of the algorithm. Thereafter, whenever energy of any cluster head becomes less than a threshold value, it sends a request to the sink node for re-invoking the algorithm. The proposed modified GSO cluster head selection algorithm contains three phases:

- Luciferin Update Phase
- Cluster Formation Phase
- Neighbourhood-Range Updation Phase

Different phases of the algorithm are described in the following subsections. The proposed modified GSO algorithm for cluster head selection is summarized as Algorithm 1.

3.2.2.1 Luciferin Update Phase

Initially, each sensor has its own luciferin intensity. The luciferin intensity of each sensor node in the modified GSO algorithm is updated depending upon its

- Present luciferin intensity $l(t)$.
- Remaining residual energy $e(t)$. The more the residual energy, more will be the luciferin value as the cluster heads consume more energy than other nodes.
- Number of neighbour nodes within its decision range $|H_t|$. The higher the connectivity index, higher will be its luciferin value as that reduces the number of required clusters.
- Voting index $v(t)$. Higher voting index is proportional to its luciferin index as it will help the selection of cluster-heads geographically distributed and hence will improve the coverage of the network.

Thus, the sensor luciferin update rule for sensor s_i is given by

$$s_i.l\,(t+1) = \frac{w \times s_i.l\,(t) + x \times \frac{s_i.e(t)}{e_0} + y \times \frac{s_i.|H_t|}{M} + z \times \frac{s_i.v(t)}{NoS}}{w+x+y+z} \tag{3.3}$$

where w, x, y, z are constants denoting weights of the factors.

Since according to luciferin index a node is voted in the voting phase, therefore, it is important to update the luciferin value in a significant way. In the above equation, four main factors have been considered.

3.2.2.2 Cluster Formation Phase

In the Modified GSO algorithm, instead of glowworm movement phase, a new phase named cluster formation phase is introduced. This phase is further divided into two sub phases (i) Voting phase and (ii) Cluster formation phase as described below.

- Voting Phase-In the find_nighbours() procedure as described in Algorithm 1, the "Hello" messages are sent to all single hop neighbours. Each node that receives a "Hello" message, sends a reply to its sender along with its unique id.

  During the cluster head selection, a voting mechanism is executed to select the cluster head with respect to every sensor node. On behalf of every node, a vote is registered for the node which has the highest luciferin value among its neighbours.In case of a tie, i.e; if more than one nodes have same and highest luciferin value then the vote is registered for the node with lowest index.

- Cluster Head Selection Phase- All the non cluster-head node n becomes the member of the cluster with cluster head k which has the maximum voting index among node n's neighbours.

Thus, in the voting phase, each node votes the node with the highest luciferin value in its neighbourhood, and voting index of each node is calculated. In the cluster formation phase, all the voted nodes are informed that they have been selected as cluster heads and also the set of nodes which have voted it become the member nodes of its respective cluster. This voting phase also prevents the overlapping of the clusters.

3.2.2.3 Neighbourhood-Range Updation Phase

In the neighbourhood range update phase, the neighbourhood range for each sensor s_i is revised according to the following rule.

$$s_i.r_d(t+1) = \min\{s_i.r_s, max\{0, s_i.r_d(t) + \alpha\,(M - |s_i.H_t|)\}\} \tag{3.4}$$

Where α is a constant parameter.

The above procedure is repeated until all the clusters have the desired number of neighbours or the clusters have become stable, that is, no changes have been found in cluster head selection phase for 3 consecutive iterations.

At the end of the execution of the algorithm, the cluster head selection information for each node is sent to them. Also, the nodes which are selected as the cluster head are notified by the sink node.

The Edge server is installed on the newly obtained cluster head.

The entire procedure is summarized as Alorithm 1.

3.2.3 Complexity Analysis

Control message overhead is an essential metric for protocol complexity analysis. In this section, we have analyzed the complexity of the proposed algorithm.

Lemma 1. The complexity of the algorithm's control messages is O(N).

Proof. Different types of message passing used in the proposed algorithm are as follows:

Hello_Message: From each node to their H one-hop neighbour.

Reply_Message: From each node to their P sender nodes of Hello_Message.

Details_Message: From each node to sink node.

Cluster_Head_Selection_Message: From sink node to each node

Number of one-hop neighbour of each node H is trivially less than or equal to N, where, N is the number of deployed nodes. Therefore, $O(H) = O(N)$.

Number of sender nodes P of hello_message is also trivially less than or equal to N. Therefore, $O(P) = O(N)$.

$\therefore O(H) + O(P) + O(N) + O(N)$, that is, $O(N) + O(N) + O(N) + O(N)$, that implies, $O(N)$.

3.3 Experimental Results

The experiments are conducted using OMNET++ network simulator to validate the proposed clustering technique. The experimental setup is described first followed by a brief discussion of the results.

3.3.1 OMNET++

It is a versatile, modular, and component-oriented object-oriented C++ library and framework designed for simulating networks, including both wired and wireless communication networks as well as queueing networks [129]. The library offers a graphical user interface built on an Eclipse-based IDE along with various additional tools (OMNET++) [130].

Algorithm 1: Modified GSO for clustering

Struct *Node* **contains**
- l(t) := Luciferin value of node at time t;
- N(id,d):= Set of all one hop neighbour id with distance d;
- r_s:= Sensing Range;
- r_d(t):= Decision Range at time t;
- e(t):= Energy level at time t;
- v(t):=voting index at time t;

end

Input: node structure details of every node at t=0

Output: selected cluster head for each node

$S := \{s_1, s_2, ...s_{NoS}\}$ *the set of sensor nodes to be clustered*;

$iter\_max :=$ *maximum number of iterations*;

$M :=$ *desired number of neighbour nodes*;

$t := 0$;

H_t:=N;

$CH = \Phi$;

while $(t \leq iter\_max)$ **do**

/* Luciferin update phase */

for *each sensor* s_i **do**

$$s_i.l\,(t+1) = \frac{w \times s_i.l(t) + x \times \frac{s_i.e(t)}{e_0} + y \times \frac{s_i.|H_t|}{M} + z \times \frac{s_i.v(t)}{NoS}}{w+x+y+z};$$

end

/* Cluster Formation phase */

for *each sensor* s_i **do**

$s_i.H_t(id, d) = find\_neighbours(s_i.r_d(t))$;

$k = find\_max\_luciferin\_node(s_i.H_t)$;

$s_k.v(t+1) = s_k.v(t) + 1$;

end

for *each sensor* $s_i \not\subseteq CH$ **do**

$k = find\_max\_voted\_node(s_i.H\,(t))$;

$s_i.ch = s_k$;

$CH = CH \bigcup s_k$;

$s_k.ch = s_k$;

end

if *No change in cluster head selection for three consecutive iterations* **then**

exit from loop ;

/* Decision range update phase */

for *each sensor* s_i **do**

$s_i.r_d(t+1) = \min\{s_i.r_s, max\{0, s_i.r_d(t) + \alpha\,(M - |s_i.H_t|)\}\}$;

end

t=t+1;

end

for *each sensor* s_i **do**

$send\_cluster\_head(s_i, s_i.ch)$;

$send\_member\_node(s_i.ch, s_i)$;

end

OMNET++ is more scalable than other simulator for large-scale IoT simulation as reported by [131].

INET Framework [132] contains models for Internet stack, wired and wireless link layer protocols that makes it suitable for IoT applications.

3.3.2 Results and Discussion

As a proof of concept, a INET-based application is developed to test the performance of the proposed algorithm.

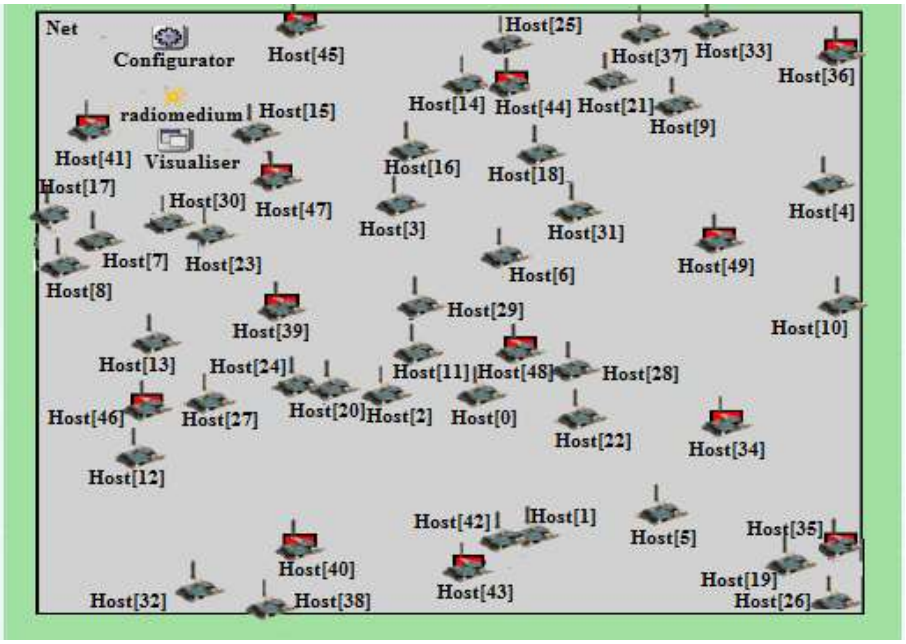


Figure 3.1: Deployment of the IoT Network in INET

Nodes are deployed randomly in the experimental region. Fig. 3.1 illustrates an example of a network deployment in the INET platform, featuring 50 nodes with the sink positioned at the center of the area. The nodes marked with rectangular boxes represent the cluster heads chosen by the modified GSO algorithm. The figure demonstrates that the cluster heads are evenly distributed across the area of interest.

Let us consider the case as shown in Fig. 3.2 , Say N1, N2, N3, N4, N5 are neighbours to each other. Suppose in the voting phase it is seen that N2 and N5 have same Luciferin value and both of them are highest among other neighbours. Then according to the algorithm if there is a tie between N2 and N5, then vote will be registered for N2 as it has

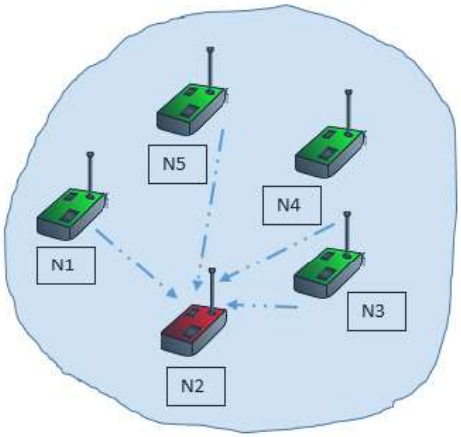


Figure 3.2: Cluster head selection in case of a tie among more than one Cluster Heads

minimum identity number.

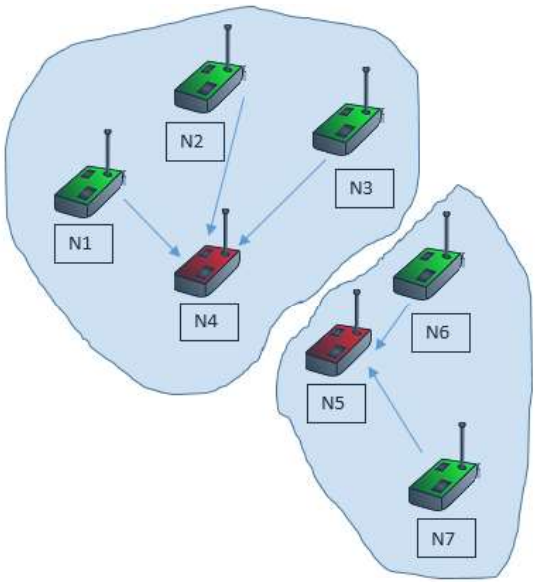


Figure 3.3: Cluster head selection in case of overlapping of clusters

Consider the following scenario as shown in Fig. 3.3. N1, N2 and N3 are neighbors of N4 and these three nodes vote N4. Node N5 has neighbor nodes N4, N6 and N7. These three nodes vote N5. According to the algorithm, since N4 is already chosen as a cluster head, it will not be any node member of any other cluster. So N1, N2, N3, N4 will form a cluster and N5, N6 and N7 will form another cluster.

After clusters are formed, each node sends data at a constant rate of 4 packets per second. Each packet can contain at most 8 kbits i.e. 32 kbits/sec. Thus, in each round of simulation, each node/the network communicates 32kbits in 1 second. the default communication range for the nodes is 70 to 100 metres following IEEE 802.15.4 standard.

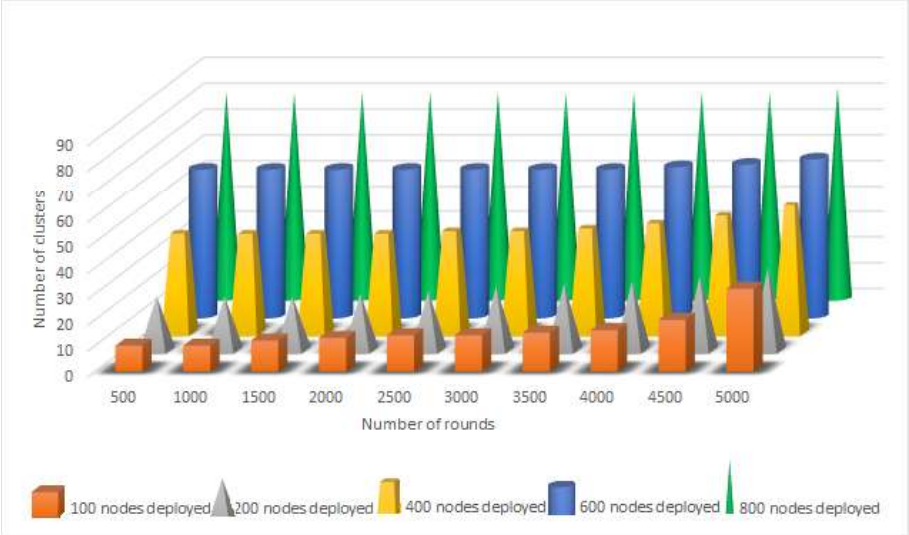


Figure 3.4: Changes in number of clusters with respect to network size and simulation time

The first experiment is conducted to explore how the stability of clusters varies with network size. From Fig. 3.4 it is observed that the clusters become more stable when more nodes are deployed. Here, by number of rounds we mean the time after deployment as data is being transmitted after a fixed interval. The proposed algorithm is invoked whenever the residual energy of the cluster head is below a threshold energy, and eventually, a new cluster head is selected. When the number of deployed nodes are more, since there is more option of re-selection of cluster head, the clusters are more stable than when the number of nodes deployed are less. A network of 600 nodes and more indicates more or less stable clustering performance.

The next experiment is conducted to explore the effect of node densities and communication ranges on cluster formation as shown in Fig. 3.5. The figure reveals that the number of clusters formed is dependent on the communication range of the deployed nodes. Communication range of the nodes and the number of cluster formed are inversely proportional to each other. As according to the proposed algorithm, only one hop neighbours can remain in the same cluster, therefore, more the communication range, less would

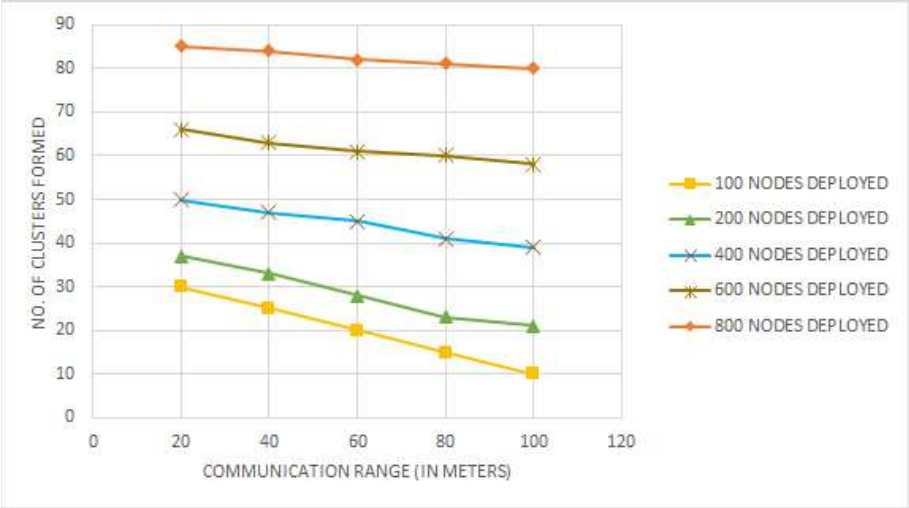


Figure 3.5: Effect of sensors' densities and communication ranges on number of clusters

be the number of clusters, as one cluster can now cover a wider geographic area.

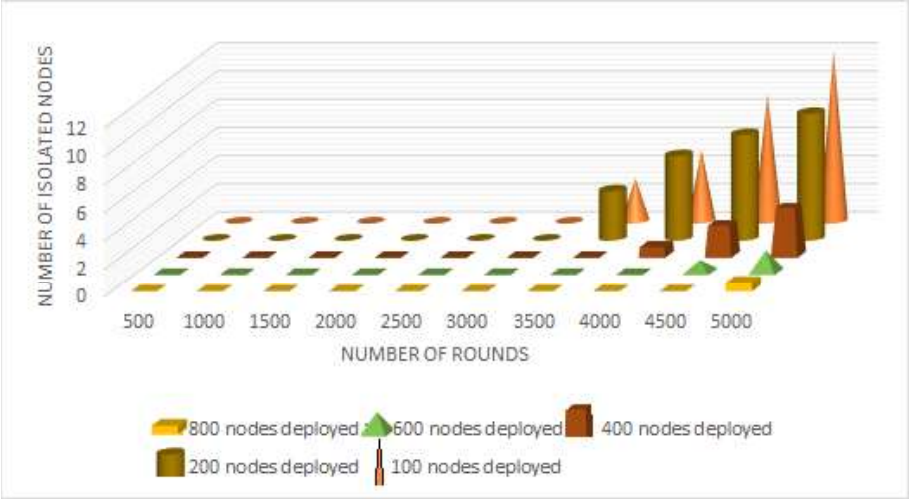


Figure 3.6: Number of isolated nodes with respect to number of rounds

The next experiment is focused to explore the relation between the number of isolated nodes and number of rounds. From Fig. 3.6, it is observed that, initially, there are no isolated nodes. However, after running for longer time, there would be some isolated nodes as many neighbour nodes would be dead. It also depicts that percentage of isolated nodes are inversely proportional to the number of nodes deployed. A node is isolated only when all its one hop neighbour are dead. For a network of 800 nodes, even after 4500 rounds, that is, 4500/4=1125 seconds after initiation (sending 4500 messages), there are hardly

any isolated nodes. Even a sparse network of 100 nodes is able to operate successfully without any isolated nodes for 3000 rounds following the proposed algorithm.

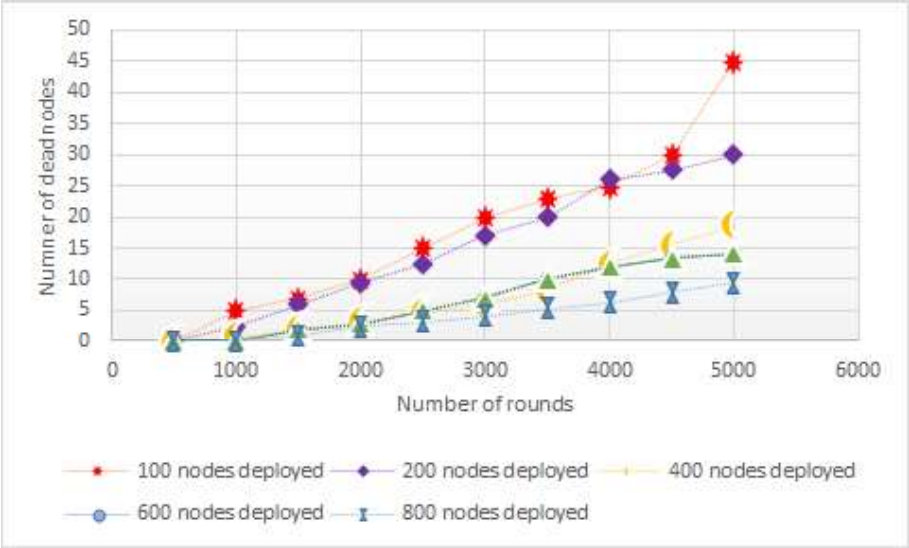


Figure 3.7: Effect of percentage of dead nodes per round on number of nodes deployed

Consequently, the next experiment is conducted to explore the relation between the percentage of dead nodes and number of rounds. It is observed from Fig.3.7 that the rate of increase in number of dead nodes is very low. Since the algorithm is using only one hop transmission, and re-selection of cluster heads ensures proper load balancing, hence, the life span of individual nodes gets improved.

Figure 3.8: Percentage of Isolated nodes with respect to varying constants

In Fig. 3.8, Fig. 3.9 and Fig. 3.10, the percentage of isolated nodes, number of clusters formed and percentage of dead nodes are tested with varying constants w, x, y

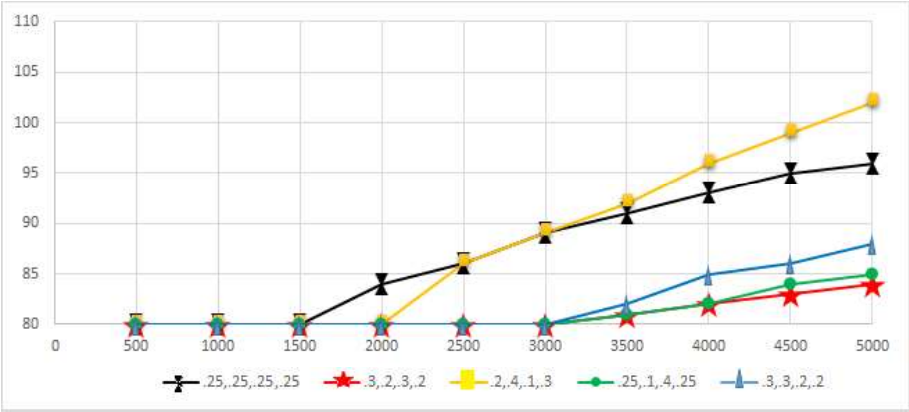


Figure 3.9: Number of clusters formed with respect to varying constants

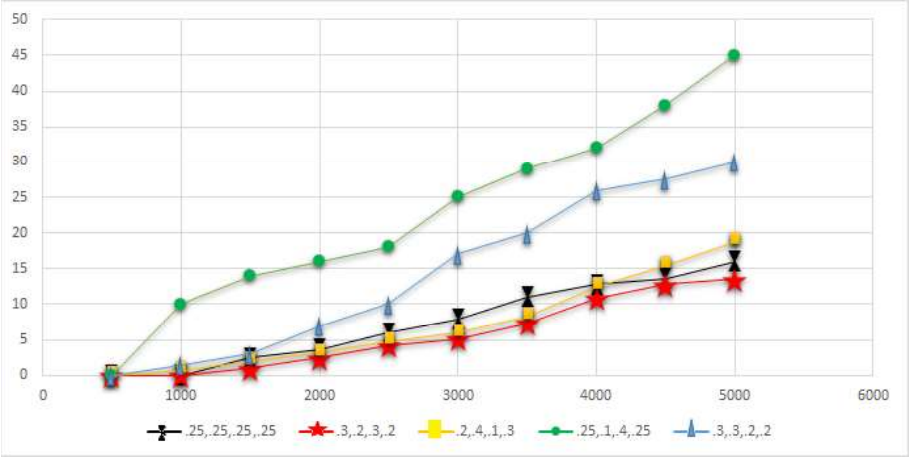


Figure 3.10: Percentage of dead nodes with respect to varying constants

and z respectively where the weight coefficient of each evaluation factor satisfies w + x + y + z = 1. Different representative value combinations are explored in order to cover the entire horizon. These constants are used in equation 3.3. As can be observed from the figures, the value combination of $w = 0.3, x = 0.2, y = 0.3, z = 0.2$ gives the best results as it results in more stable clusters (Fig. 3.9) resulting in lesser percentage of dead nodes (Fig. 3.10). Thus, remaining energy of the nodes should be given more weight while calculating the fitness value.

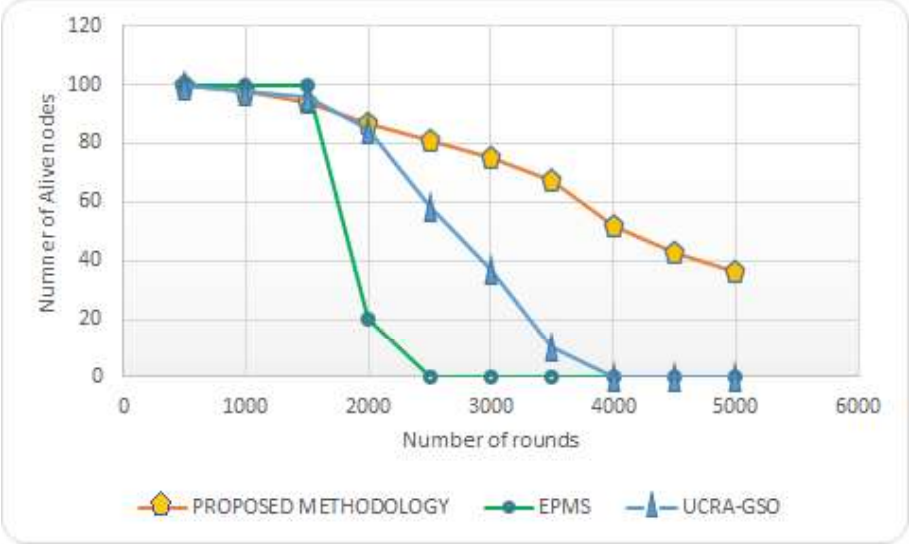


Figure 3.11: Comparison of proposed methodology with EPMS and UCRA-GSO with respect to number of alive nodes per round

To compare the effectiveness of our protocol, our work is compared with the EPMS protocol [84] and UCRA-GSO protocol [85]. In EPMS, the authors have used Particle Swarm Optimization and in UCRA-GSO authors have used GSO for clustering. The comparison is made on the basis of the number of nodes alive per round. The results are depicted in Fig. 3.11. From the figure it is clear that, though in initial rounds the EPMS and UCRA-GSO are found to give better results, but, after some nodes are dead, the proposed mechanism works better as it is changing the cluster heads depending on the residual energy. The energy consumed by the nodes to rerun the algorithm is already taken care of by the simulation environment. Though it is obvious that more energy is consumed while re-selecting the cluster heads, but it is taken into account that the reselection is done only when the cluster head has not the remaining residual energy to

perform as a cluster head, and in that scenario the re-selection increases the lifetime of the network by balancing the load among nodes. Because of this, load balancing among the number of alive nodes is better in our protocol.

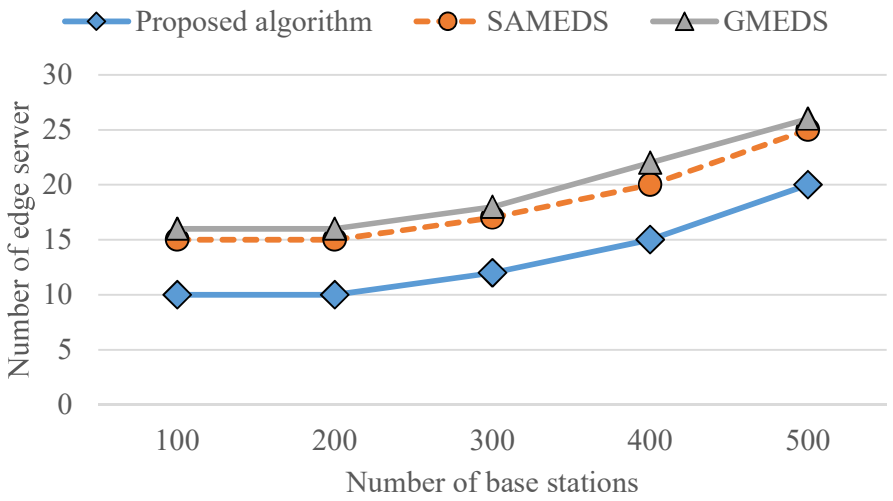


Figure 3.12: Comparison among different edge server placement algorithms w.r.t number of edge servers deployed vs number of base stations

Here, the performance of the proposed method after deploying the edge nodes in the positions of selected cluster-heads is evaluated against two existing approaches GMEDS and SAMEDS [56] in terms of the count of edge servers required for varying the count of base stations. Fig. 3.12 demonstrates that all three curves exhibit a growing tendency as the network size grows. However, the proposed technique is found to outperform the state-of-the-art approaches. This is because the straw polling phase of the proposed technique enables to place the edge servers in a geographically distributed manner. Further, the edge server selection phase ensures the non-overlapping of clusters.

Thus, to summarize our findings, the number of clusters are found to become stable for a long runtime if more than 600 nodes are deployed. More communication range would obviously call for lesser no of clusters. Denser networks provide better performance. With 800 nodes, the network performs better without any isolated nodes even after 4500 rounds.

3.4 Summary

In this chapter we have introduced an adaptive cluster head selection algorithm based on Glow-worm swarm optimization. The algorithm is able to divide the whole network into optimised number of clusters. In the developed technique the number of nodes in each clusters is not fixed, it automatically changes depending on the number of alive nodes in the network, which reduces the number of isolated nodes and also increases the lifetime of the network. The algorithm ensures minimum overlapping of the clusters using voting mechanism which minimizes the communication overhead. The algorithm is automatically initiated by the sink node when any cluster-head is dead. Repeated selection of cluster heads and reformation of the clusters ensure proper load balancing in the network and simultaneously increases the lifetime of the network. The designed algorithm is executed in OMNET++ and have shown the various outcomes in different conditions such as after different number of rounds, changing the sensor densities, communication ranges and varying the constants. The results show how the clusters are self-adaptable after different number of rounds depending on number of alive nodes. In the proposed methodology, the edge server placement in an IoT-edge computing scenario is examined as a constraint optimization issue. A modified Glowworm Swarm Optimization algorithm is introduced for identifying the placement site closer to the selected base stations. The proposed method finds a balance among the operational demands of the edge server, and the no of the base stations covered by an edge. Moreover, it has also been demonstrated how the developed technique works better than a state-of-the-art technology in terms number of alive nodes after a certain number of rounds.

After deploying the edge nodes in the optimum positions it is found that to improve overall service efficiency and reduce task offloading delays, it is necessary to propose an effective task offloading approach for managing IoT-edge computing resources and processing computation workloads for deadline-sensitive applications. In the next chapter we have focused on this issue.

List Of Publications

1. **Journal**:

   M.Bakshi, C.Chowdhury and U.Maulik, "Energy-efficient cluster head selection

algorithm for IoT using modified glow-worm swarm optimization," *The Journal of Supercomputing*, Springer, 2021, DOI: https://doi.org/10.1007/s11227-020-03536-z, 77(7), pp. 6457-6475.

2. **Conference**:

   M.Bakshi, M.Roy,U.Maulik and C. Chowdhury, "An optimal edge server placement algorithm based on glowworm swarm optimization technique ," in *4th International conference on frontiers in computing and systems*, Mandi, Himachal Pradesh, 2023.

Chapter 4

Deadline-Aware Data-Centric Offloading Algorithm

4.1 Introduction

In this chapter, an efficient job offloading approach is proposed to enhance service efficiency and minimize delays in offloading tasks for IoT-edge computing. The focus is on managing resources and processing workloads for time-sensitive applications. The key highlight is proposing a job offloading algorithm based on Cuckoo search optimization for IoT edge environments to reduce job failure rates of time-sensitive tasks. Tasks can be processed on the device itself or offloaded to another device within the same or different cluster, based on factors like deadlines, data requirements, and CPU cycle needs. The use of meta-heuristic optimization algorithms is crucial due to the complexity of the decision-making process. The Cuckoo search-based optimization technique proves effective in achieving convergence and utilizing levy flights for exploration. The proposed Cuckoo search-based job offloading approach (TOCS) addresses the requirements efficiently and devises an optimal offloading strategy for edge devices.

An overview of this technique is detailed in the following section.

4.1.1 Overview of Cuckoo Search Algorithm (CSA)

CSA is one of the latest nature-inspired meta heuristic algorithms, developed in 2009 by Xin-she Yang and Suash Deb [133]. CS is based on the obligate brood parasitic behavior of some Cuckoo species in combination with the Levy flight behaviour of Cuckoo bird. The pseudocode for Cuckoo search is described in Algorithm 2.

Algorithm 2: CSA

```
Objective function f(x);
Generate initial population of n host nest;
Evaluate fitness and rank eggs;
while t > MaxGenerationorStopcriterion do
    t=t+1;
    Get a cuckoo randomly/ generate new solution by Levy flights;
    Evaluate quality/fitness F_i;
    Choose a random nest j;
    if (F_i > F_j) then
        Replace j by the new solution;
    end
    Abandon a fraction (pa) of worse nests;
    build new ones at new locations via Levy flights;
    Keep the best solutions (or nests with quality solutions);
    Rank the solutions and find the current best;
end
Post process results and visualization
```

When generating new solutions $x_{\alpha}^{(t+1)}$ for αth cuckoo, a Levy flight is performed using the following equation

$$x_{\alpha}^{(t+1)} = x_{\alpha}^{t} + \alpha s \bigotimes H(p_a - \varepsilon) \bigotimes (x_j^t - x_k^t) \tag{4.1}$$

where x_j^t and x_k^t are two different solutions selected randomly by random permutation, H(u) is a Heaviside function, ε is a random number, and s is the step size.

On the other hand, the global random walk is carried out by using Levy flights

$$x_{t+1}^{i} = x_t^i + \alpha L(s, \lambda) \tag{4.2}$$

$$L(s, \lambda) = \frac{\lambda \Gamma(\lambda) sin(\pi\lambda/2)}{\pi} \left(\frac{1}{s^{1+\lambda}} \right), (s \geqslant s_0 > 0) \tag{4.3}$$

where $\Gamma(\lambda)$ is the Γ function and λ the random step length. Levy flights essentially provide

Table 4.1: Rules of Cuckoo Search

Rule No	Rule	How it helps in solving the optimization problem
i	Each cuckoo lays one egg at a time, and dumps its egg in a randomly chosen nest.	The new cuckoo egg intends to replace a previous egg by improving the quality of solutions.
ii	The best nests with high quality of eggs will carry over the next generation.	The "selection of best" finds the best local optimum among the cuckoo population at each generation, progressing the convergence of the cuckoo to a feasible global optimum.
iii	Each cuckoo egg laid in a host nest has an assigned discovery rate value, p_a, and when a predefined discovery condition is met, the host bird can either build a new nest or just get rid of the cuckoo egg.	A completely new solution far from current population's solution is generated enabling the global scale exploration.

a random walk while the random steps are drawn from a Levy distribution for large steps $(1 < \lambda \leq 3)$.

The three idealized rules of Cuckoo search and how these rules help in solving the optimization problem are listed in Table 4.1.

CSA has mainly two advantages

i. It satisfies the global convergence requirements.

ii. It supports local and global search capabilities.

Recent studies show that in comparison with Genetic Algorithm, Cuckoo Search has a high exploration ability due to its mutation related Levy flights, yet can converge quickly for job scheduling at grids as in [134].CSA has two distinct advantages, such as, efficient random walks and balanced mixing. Levy flights are reported to be applied in optimization to improve the efficiency of the search process for nature inspired algorithms as in [135]. In [136], authors have proved that CSA can degenerate into a variant of Differential Evolution (DE), as well as an Accelerate Paticle Swarm Optimization (APSO) algorithm. This means that SA, DE and APSO are special cases of CSA.

4.2 Proposed Methodology

The problem is formally defined by presenting the architecture and system model first. This section then goes into detail about the suggested job offloading strategy.

4.2.1 System Model

The IoT network consists of a number of distributed IoT nodes $IoT_1, IoT_2, \ldots IoT_n$ where each IoT node generates a single job at a time. Each IoT node holds different parameters described in Struct IoT (Section 4.2.1).

Struct IoT **contains**

> IoT_{id} `// Unique identity of each IoT`
>
> PoS(x,y) `// Position of the IoT node`
>
> $Edge_{id}$`// Controlled by Edge node`
>
> avail=1`// Number of jobs can be alloted=1`

end

IoT nodes produces at most n jobs at a Time denoted by $job_1, job_2, \ldots.., job_n$.

Struct job **contains**

> job_{id} `// Unique identity of each job`
>
> CPUCycleRequired `// Assumption:1 instruction executes per CPU cycle`
>
> TimeDeadline`// Maximum time unit allowed for execution`
>
> IoT_{id}`// Unique Identity of the IoT node where job is alloted`

end

Edge layer of the network consists of Edge nodes $Edge_1, Edge_2, \ldots., Edge_m$ where $m << n$. One Edge node controls the IoT nodes within one hop distance. The Edge node under whose control the IoT node is , is known as its OwnECU, other edge nodes are defined as its "OtherECU".

Struct $Edge$ **contains**

> $Edge_{id}$ `// Unique identity of each Edge`
>
> PoS(x,y) `// Position of the edge node`
>
> avail`// Number of jobs can be alloted at a time`

end

The job is executed successfully if it can be executed by an available resource (that is by the IoT device, its OwnECU or OtherECU) and is completed before the Time Deadline assigned to it. The objective of the work is to provide a job offloading strategy applicable in heterogeneous edge server environment that will provide the following optimization

1. minimize the response time
2. minimize energy consumption
3. Maximize resource utilization
4. maximize the percentage of successful job execution while considering the data-centric heterogeneous job with CPU-demand and deadline awareness.

4.2.2 Algorithm Description

In this section the methodology of the proposed work is described. The significance of all the constant terms used in the methodology is described in Table 4.2.

Table 4.2: Significance of the constant terms

Symbol	Significance
α	CPU Cycle Per Unit Time for IoT
β	Energy required by IoT device per unit time
γ	Bandwidth
δ	CPU Cycle per Unit Time for ECU
ϵ	required per unit time for transferring data
ζ	Energy required by ECU device per unit time

If the job_j is executed in an IoT the Time required is dependent of number of CPU cycle needed for execution of the job and CPU Cycle Per Unit Time for the IoT device and is calculated as

$$T = \frac{job_j.CPUCycleRequired}{\alpha} \tag{4.4}$$

The Energy required is calculated as follows.

$$E = T \times \beta \tag{4.5}$$

If the job is executed in an ECU, the Time required, is dependent on

1. The data transfer time from the IoT device to the ECU
2. The amount of data to be transferred
3. Number of CPU cycle needed by the job
4. CPU Cycle Per Unit Time for the ECU device

The data Transfer time for job_j for executing in $resource_i$ is calculated as

$$T_1 = \frac{job_j.DataNeeded}{\gamma} \times \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{4.6}$$

where $(x_1, y_1) = job_j.IoTid.PoS$ and $(x_2, y_2) = resource_i.PoS$

The time required for execution of the job_j in the edge device is calculated as

$$T_2 = \frac{job_j.CPUCycleRequired}{\delta} \tag{4.7}$$

The total time required for execution of a job by an ECU is calculated as

$$T = T_1 + T_2 \tag{4.8}$$

The total energy consumed while executing a job through an ECU device is calculated as

$$E = T_1 \times \epsilon + T_2 \times \zeta \tag{4.9}$$

The normalised Time for executing a job is calculated as

$$NT = \frac{T}{PTT} \tag{4.10}$$

Where T is the total time needed for job execution and PTT is the total time needed for

executing all jobs.

The normalised Energy consumed required for executing a job is calculated as

$$NE = \frac{E}{PTE} \tag{4.11}$$

Where E is energy consumed for execution of the job and PTE is the total energy consumed for executing all jobs.

The normalised resource utilization for $resource_i$ is calculated as

$$R = \frac{PR[resource_i] + 1}{resource_i.avail} \tag{4.12}$$

Where $PR[resource_i]$ is denoting the number of VMs used by the ECU where the job is being executed and $recource_i.avail$ is denoting the total number of VMs available in that ECU.

The fitness value of the solution point (job, resource used) is calculated using

$$fitval = C \times \frac{(\chi \times (1 - NE) + \psi \times (1 - NT) + \omega \times (R))}{(\chi + \psi + \omega)} \tag{4.13}$$

Where χ, ψ, ω are constants denoting weights of the factors. c=1 if the job can be completed successfully using the referred resource, otherwise c=0 and are constants χ, ψ, ω denoting weights of the factors.

The proposed cuckoo search based job offloading (TOCS) algorithm (algorithm 3), takes the list of jobs, IoTs and ECUs deployed and returns the job offloading decision that is which job will be executed in the IoT itself, and which will be executed in its own ECU or other ECU. The decision is taken in such a manner such that the overall Time for execution and energy consumed is minimised, whereas number of successful executed job and resource utilization is maximised.

Since there are n number of IoTs and it is assumed that each IoT device is generating a single job therefore there are n jobs to process. M number of ECUs are deployed, therefore $n^{(m+n)}$ number of possible solutions are available to place the n number of jobs in n IoTs

Algorithm 3: TOCS

Input: $jobDetails = [job_1, job_2, ..., job_n]$;
$IoTDetails = [IoT_1, IoT_2, IoT_3, ..., IoT_n]$;
$EdgeDetails = [Edge_1, Edge_2, ..., Edge_m]$;
Output: Optimized job offloading details;
$JR = [(job_1, r_1, Type), (job_2, r_2, Type), ..., (job_n, r_n, Type)]$;
where $job_i \in jobDetails$;
$r_i \in IoTDetails \cup EdgeDetails$;
$Type = IoT$ or $OwnECU$ or $OtherECU$

Set of all solution points
$SR = [(job_i, r_i, Type), \forall job_i \in jobDetails, r_i \in IoTDetails \cup EdgeDetails]$;
Set $JR = [(job_1, IoT_1, IoT), (job_2, IoT_2, IoT), ..., (job_n, IoT_n, IoT)]$;
foreach $JR_i \in JR$ **do**
    $JR_i.FitnessVal = F(JR_i)$;
end
((PTE,PTT,PTC,PR)=TECcalc(JR);
while *(t<MaxGeneration) or (stop criterion)* **do**
    Get a Cuckoo $SR_k \in SR$ randomly via Levy flights;
    $SR_k.FitnessVal = F(SR_k)$;
    Find $JR_m \in JR \mid JR_m.job = SR_k.job$ **if** $(SR_k.FitnessVal >= JR_m.FitnessVal)$ **then**
        TempJR=JR;
        $TempJR_m = SR_k$;
        (TE,TT,TC,R)=TECalc(TempJR);
        if *TC<=PTc and (TT<PTT or TE<PTE)* **then**
            Replace JR_m by SR_k in JR;
            PTT=TT;
            PTC=TC;
            PTE=TE;
            PR=R;
        end
    end
    Abandon a fraction (pa) of worse nests from JR;
    build new ones at new locations in JR via Levy flights;
    Keep the best solutions (or nests with quality solutions) in JR;
    Rank the solutions of JR and find the current best;
end

```
Procedure CheckCompatibility(job_j,resource_i,Type)
    if Type==IoT then
        Calculate T using Equation 4.4;
        Calculate E using Equation 4.5
    end
    else
        Calculate T_1 using Equation 4.6;
        Calculate T_2 using Equation 4.7;
        Calculate T using Equation 4.8;
        Calculate E using Equation 4.9;
    end
    if PR[resource_i] < resource_i.avail and T < job_j.TimeDeadline then
        comp = 1;
    end
    else
        comp = 0;
    end
    return (E,T,comp);
```

```
Procedure F(job_j,resource_i,Type)
    (E,T,C)=CheckCompatibality(job_j,resource_i,Type);
    NT is calculated using Equation4.10;
    NE is calculated using Equation4.11;
    R is calculated using Equation4.12;
    fitval is calculated using Equation4.13;
    return fitval;
```

```
Procedure TECcalc(JR)
    TT=0;
    TE=0;
    TC=0;
    Resource = {IoT_1 : 0, IoT_2 : 0, ...., IoT_n : 0, ECU_1 : 0, ECU_2 : 0, ..., ECU_m : 0};
    for All entries(T_i, R_i, Type_i) in JR do
        (E_i, T_i, C_i)=CheckCompatibility(T_i, R_i, Type_i);
        TE = TE + E_i;
        TT = TT + T_i;
        TC = TC + C_i;
        Resource[R_i] = Resource[R_i] + 1;
    end
    return TE,TT,TC,Resource;
```

and m ECUs.

The suitability of deciding a job to be executed in a resource is depended on the following factors:

1. Availability of the resource

2. The job execution can be completed within the given deadline

Initially all jobs are assumed to be executed in their respective IoT devices. Accordingly the solution points are selected and fitness value of all the solution points are calculated calling Procedure F(job_j,$resource_i$,$Type$) .

The compatibility of the job w.r.t the resource assigned is checked using procedure CheckCompatibility(job_j,$resource_i$,$Type$).

The Total Energy, Total Time, Total cancelation of jobs and Resource Utilization is calculated using procedure TECcalc(JR) .

A cuckoo, i.e. a solution point SR_k with job_k is randomly selected via Levy flight from the set of all solution points. Fitness value of SR_k is calculated. If the fitness value of SR_k is greater than or equals to the fitness value of already selected solution point of same job (ie. If it is local optimised) then Total Energy, Total Time, Total cancelation of jobs and Resource Utilization are calculated w.r.t SR_k. If by replacing already selected job w.r.t job_k with SR_k gives less cancellation of job, and gives better result for energy consumption, required time and resource utilization, then the already selected solution is replaced by SR_k. The process is repeated till upto three iterations no change in list of selected solution points or maximum number of iterations achieved.

4.2.3 Complexity Analysis

Let us consider the original CS algorithm's complexity before estimating the complexity of the suggested TOCS. The original CS algorithm's computational complexity was found to be $O(n \times D \times t_{max})$ [133] when estimating the population size as n of a D-dimensional problem with tmax as the maximum number of iterations. This estimate is based on the fact that O(D) operations must be performed for each population, and that the computational complexity for a population size of n is determined by the dimension

size D. Therefore, $O(n \times D)$ gives the complexity for the entire population. This complexity is only given for a single iteration; however, the computational burden increases to $O(n \times D \times t_{max})$ when the iteration size is increased to t_{max}. This represents the fundamental CS algorithm's computational complexity. Now that the proposed TOCS algorithm and the steps of the original CS algorithm are compared, it is clear that there is no additional computational load because the new TOCS retains all of its fundamental parameters. However, by using the check_compatibility() function, the computational cost will be decreased in this case as the population size decreases. If we designate this smaller population as $n_{reduced}$, then $O(n_{reduced} \times D \times t_{max})$ represents the overall reduced computational load. There is only one parameter that affects the computational complexity, and that is the population size. As a result, the suggested TOCS has lower time and space complexities than the original CS algorithm.

4.3 Experiment Results and Discussion

In this section, experimental results are presented to evaluate the performance of the proposed TOCS algorithm 3.In order to validate the results, the proposed algorithm is simulated using the CISCO PACKET TRACER simulator [1] while the code was written in Python language. In Table 4.3, the default parameter settings for evaluating TOCS are listed. The values are taken following the datasheet [137, 138].

We first discuss the intricacies of the proposed algorithm with respect to an example use case. Fig. 4.1 presents a sample deployment of 50 IoT devices with 5 ECU nodes. From the figure it is observed that after executing the proposed algorithm, some of the jobs are executed at the respective IoT devices whereas others are optimally offloaded to different ECUs.

In Table 4.3, the default parameter settings for evaluating TOCS are listed. The values are taken following the datasheet [137, 138].

We have considered the dataset of nearly 50k jobs available in [139]. For the demonstration, we have considered a set of 15K job details and obtained the results.

[1] https://www.netacad.com/courses/packet-tracer

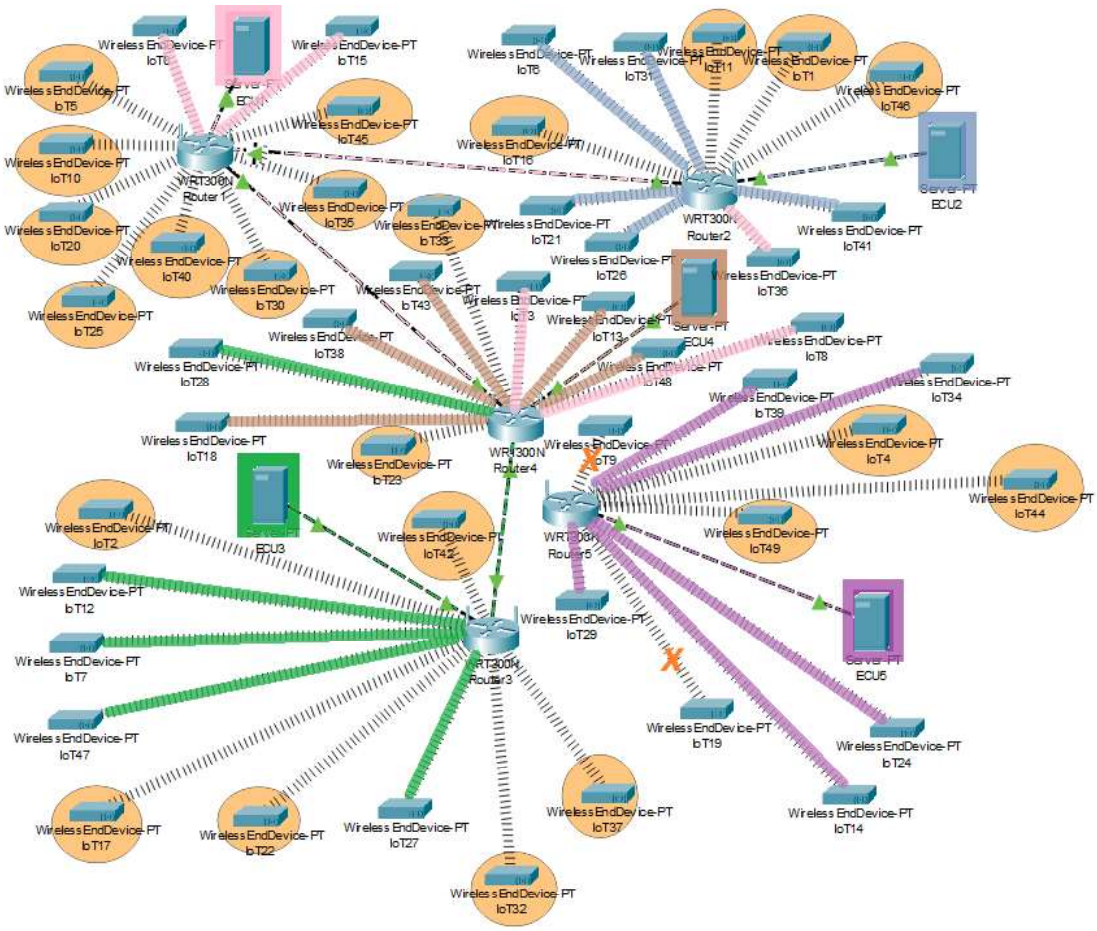


Figure 4.1: Deployment of the Network in CISCO Packet Tracer

Table 4.3: Experimental Setup Parameters

KEY PARAMETER	VALUE
n	50-900
m	4-10
VMi	[3,7]
α	16MIPS
β	50W
γ	36Mbps
δ	18000MIPS
ϵ	250kbps
ζ	300W

4.3.1 Case Study

A case study has been performed with 50 jobs. The input jobs are chosen representing different practical scenarios as follows.

1. Soft deadline with higher CPU cycle needed
2. Hard deadline with higher CPU cycle needed
3. Less data with hard deadline
4. Moderate data needed with moderate deadline
5. Less CPU cycle with soft deadline

Different aspects of the outputs obtained after the experiments are presented below. Since the main motive of the approach is to provide the optimized result at first the job cancellation percentage is checked.

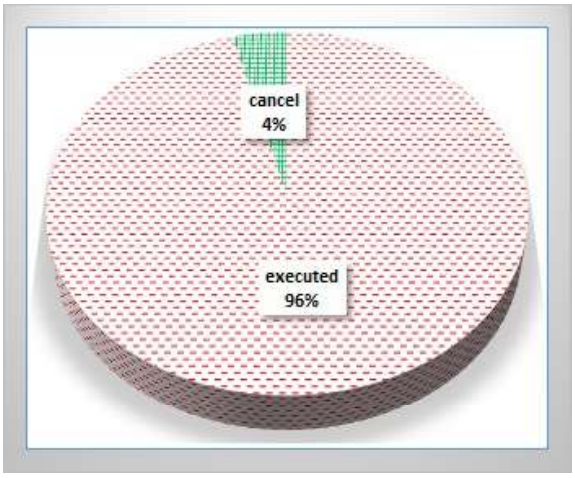


Figure 4.2: Percentage of job execution of 50 sample jobs taken for case study

Fig 4.2 shows that after job-offloading 96% jobs have been successfully executed. From the input metrics of the sample jobs it has been found that the jobs having hard deadline but more CPU cycle need are canceled as it was not possible to execute the jobs any how within the deadline. In Fig 4.3 a more detailed decription has been shown. Among the 96% successfully executed job, 46% jobs has been executed in the same IoT itself. In case of other jobs IoT devices were not cabable of providing CPU resources within the needed

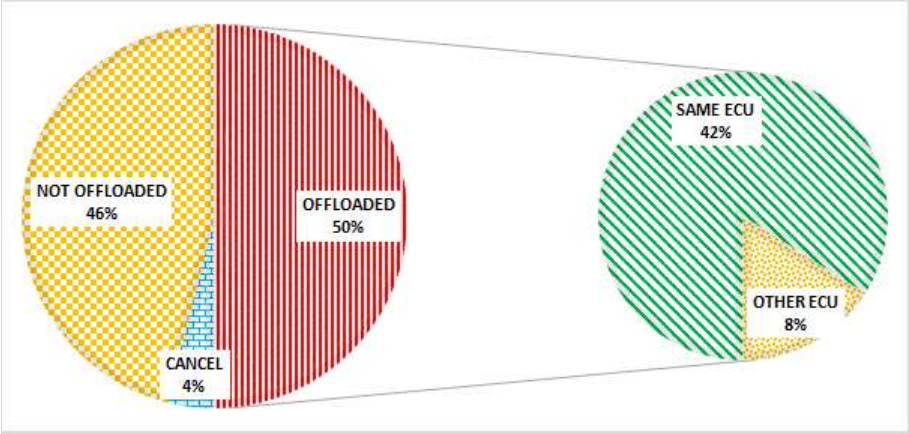


Figure 4.3: job offloading details of 50 sample jobs taken for case study

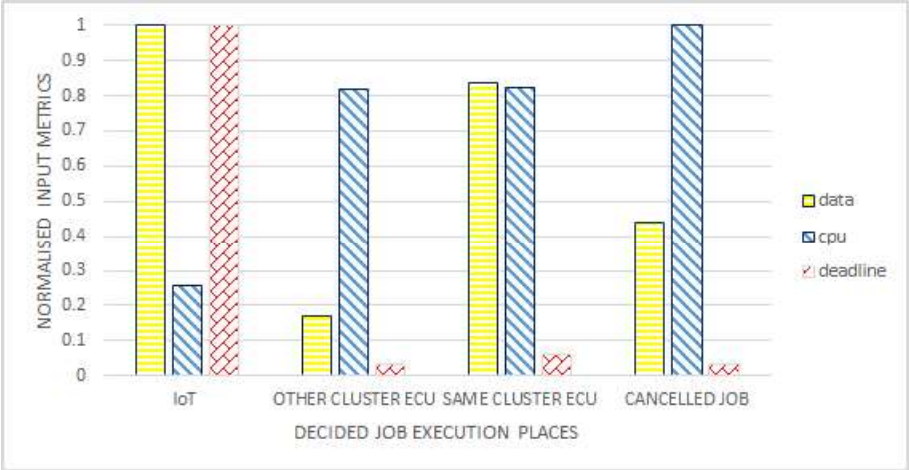


Figure 4.4: Relation between input metrics and job offloading decision

deadline. According to the availability of ECUs , 42% jobs are executed in the ECU of the same cluster whereas 8% of the jobs are executed in the ECUs of the different clusters. From the input metrics of the jobs it has been found the jobs having more deadline time allocated and lesser CPU cycle needed are able to be executed in the IoT devices itself, as IoT devices have very less processing power. The jobs that need more CPU cycle are offloaded to ECU devices. If the same cluster ECU is not available for jobs , then the job with lesser data transfer need is offloaded to other cluster ECU, to minimise the cost. Fig 4.4 is depicting the described the relation between the input metrics and job offloading decision.

In Fig 4.5 the job distribution details has been shown. It is seen that in 5 ECUs the

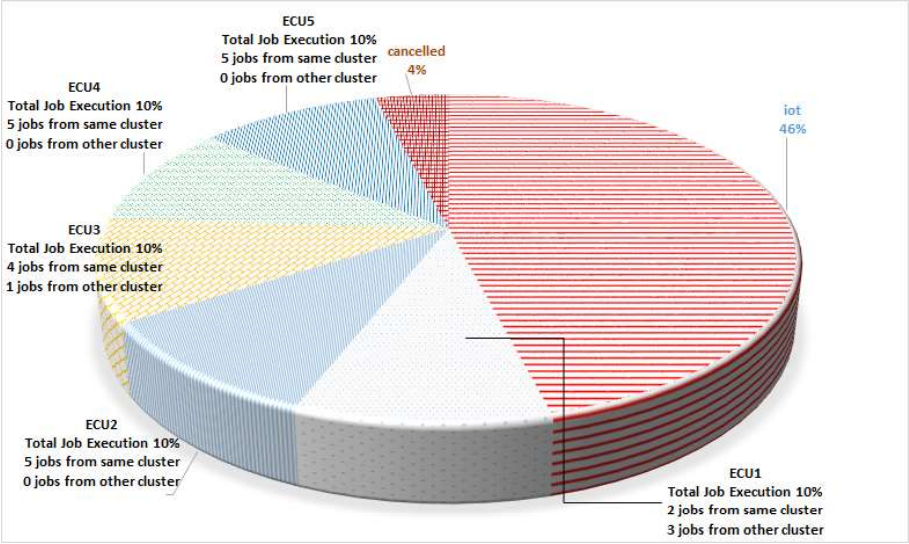


Figure 4.5: Job distribution details after offloading decision on 50 sample jobs taken for case study

workload is evenly distributed according to the proposed algorithm, though the number of jobs from same cluster and other clusters are different. The ECUs who are executing lesser jobs from same cluster are used to execute the other cluster jobs to provide the evenly job distribution in all the ECUs.

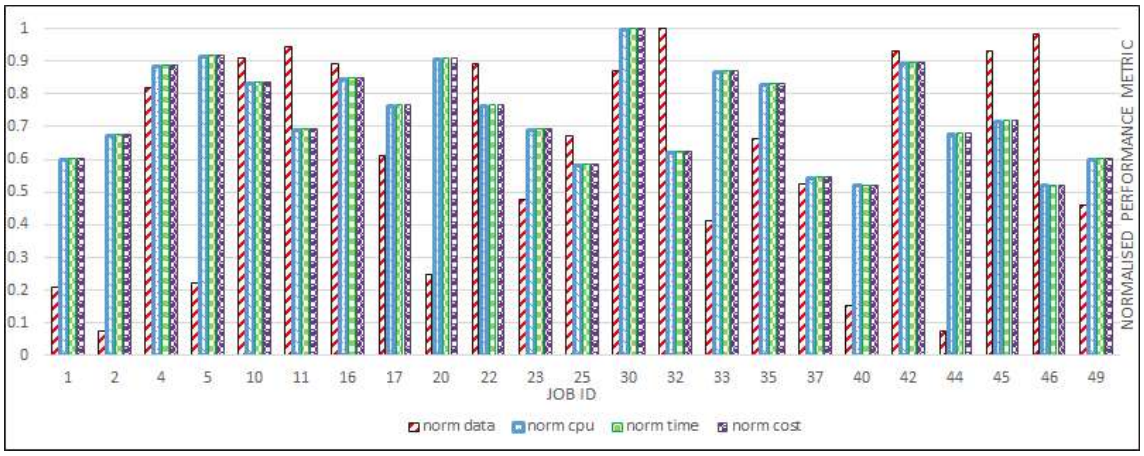


Figure 4.6: Variation of job execution cost in an IoT device subject to different parameters

The effect of different input parameters on the cost is detailed in Fig 4.6. It indicates the normalized value for data needed, CPU cycle needed, time and cost incurred for job completion for the jobs executed in the IoT device itself. From the figure it is clear that the cost and time incurred is directly proportional to CPU cycle needed and not associated with data needed. As no extra time or cost is incurred for data needed in execution, when

the job is executed on the IoT device itself.

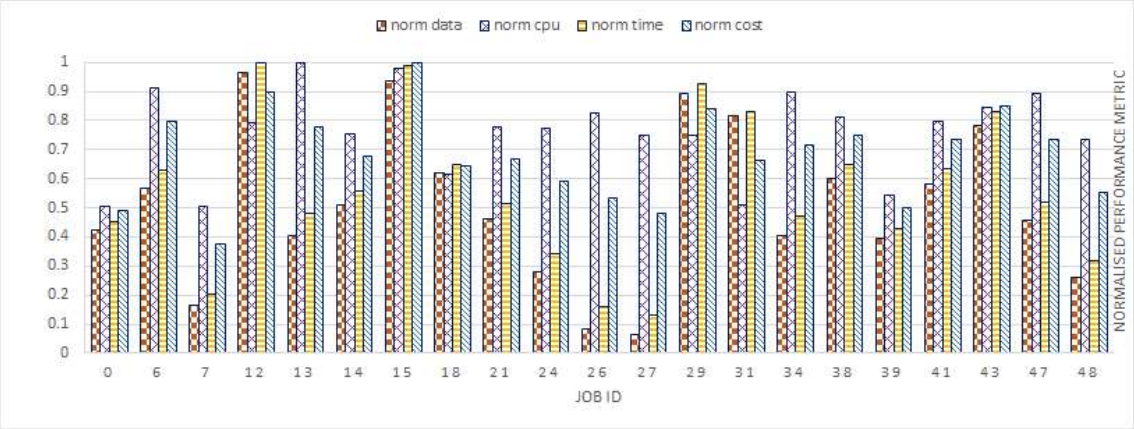


Figure 4.7: Variation of job execution cost in an ECU device located in the same cluster of the job generating IoT device subject to different parameters

Fig 4.7 indicates the normalized value for data needed, CPU cycle needed, time and cost incurred for job completion for the jobs executed in the ECU device in the same cluster as the IoT device. From the figure it is clear that the time incurred is directly proportional to data needed as the data is to be transferred from the IoT device to the ECU device. The cost incurred is dependent on both the data and CPU cycle needed, but more on CPU cycle needed for execution. Similar observation is reported for the jobs executed on ECUs of different cluster as shown in Fig 4.8.

A comparison of different metrics is reported in Fig 4.9. The metrics considered are cost-data ratio, cost-CPU cycle ratio, and time-CPU cycle ratio for various computing resources -IoT, ECU of same cluster, and ECU of different cluster. From Fig 4.9 it is observed that the time-CPU cycle ratio and cost-CPU cycle ratio are maximum when the job is executed in an IoT devices. The cost-data needed ratio is maximum when the job has to be executed in an ECU in other cluster as the cost for sending data to the ECU is maximum in this case. Considering the limited ECU resources availability, the proposed algorithm is found to achieve appreciable performance resulting in minimal job cancellation.

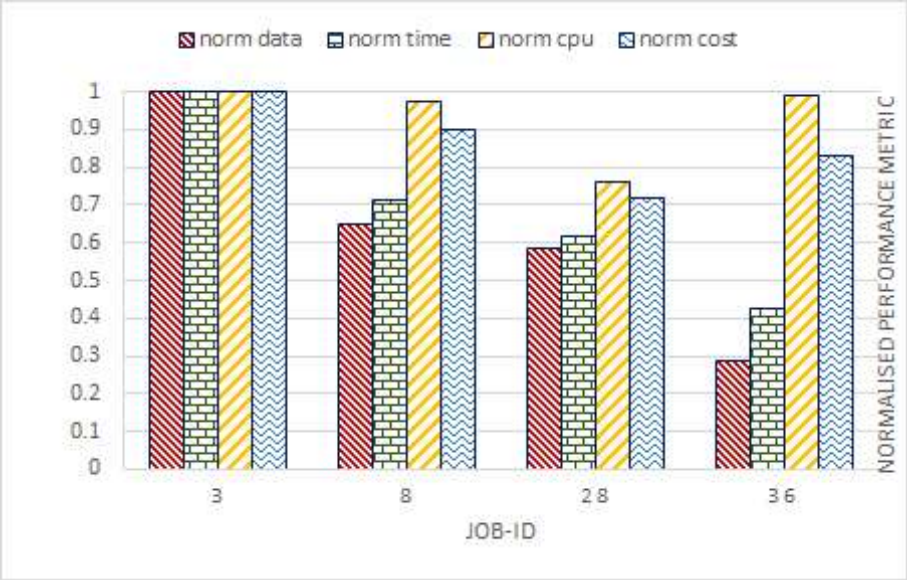


Figure 4.8: Variation of job execution cost in an ECU device located in the other cluster of the job generating IoT device subject to different parameters

4.3.2 Result Analysis in homogeneous and heterogeneous deployment

In this section the performance of the algorithm has been tested for various homogeneous and heterogeneous deployment environment. Homogeneous environment signifies that the ECUs consist of same number of VMs and vice versa.

An experiment has been conducted where a number of homogeneous ECUs have been deployed with different number of VMs per ECU. The number of cancelled task is observed and reported in Fig 4.10. It could be seen that a certain number of ECU is required, and if the VMs per ECU is increased then number of task cancellation is decreased. For lesser number of ECU, if number of VMs per ECU is increased it does not decrease the number of task cancellation.

In the next experiment, heterogeneous ECU has been deployed. Fig 4.11 is reflecting if number of VMs is more in certain ECU and lesser in certain ECU, it doesn't help in decrease of number of job cancellation. Homogeneous ECU deployment gives etter result w.r.t decreased number of job cancellation.

Figure 4.9: Comparison of different metrics in various resources

4.3.3 Performance tuning of the proposed CSJS algorithm

The default values of the setup parameters taken in this section are listed in Table 4.3. It is important to find out the optimized value of the weight parameters χ,ψ and ω used in Algorithm 3 for calculating the optimized job schedule. In Fig 4.12, the normalised time consumed, in Fig 4.13 normalised energy consumed and in Fig4.14 cancelled job to received job ratio with respect to different number of jobs are tested with varying the values of (χ, ψ, ω), respectively, where the weight coefficient of each evaluation factor satisfies $\chi + \psi + \omega = 1$.

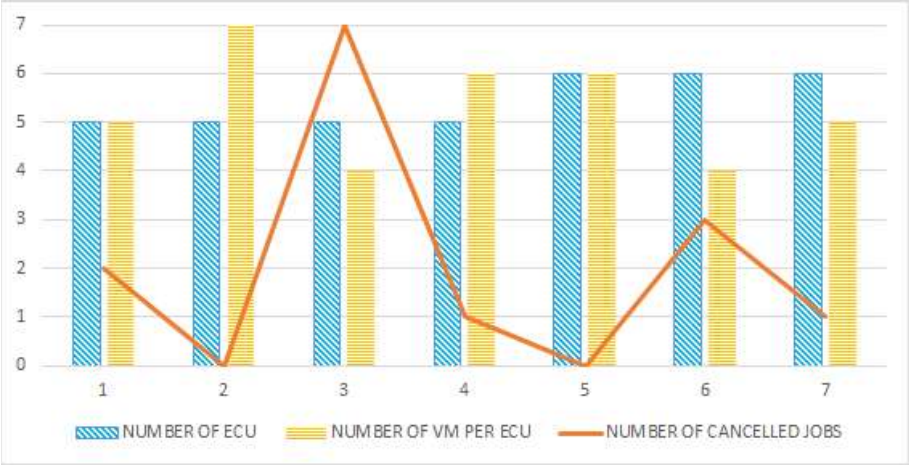


Figure 4.10: Number of cancelled jobs w.r.t various number of homogeneous ECU deployment

Figure 4.11: Number of cancelled jobs w.r.t various number of heterogeneous ECU deployment

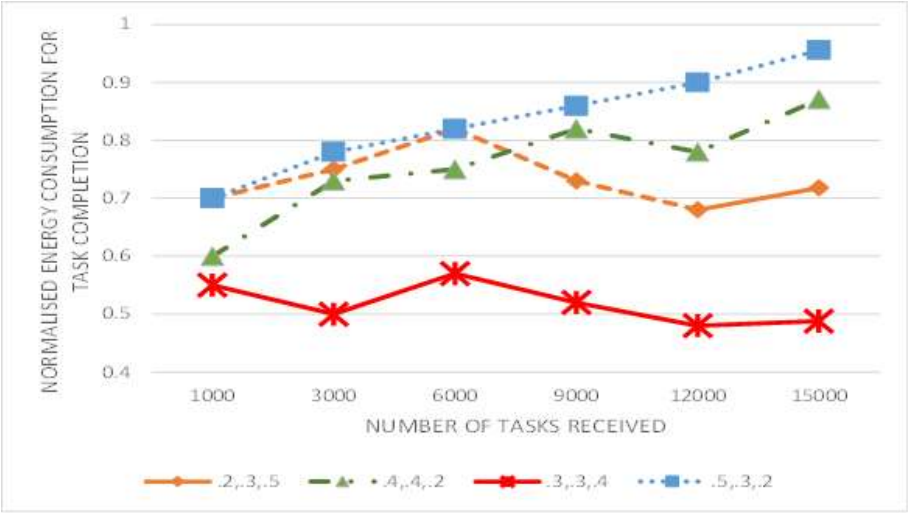


Figure 4.14: Cancelled job and received job ratio w.r.t varying constants

Figure 4.12: Normalised time consumption w.r.t varying constants

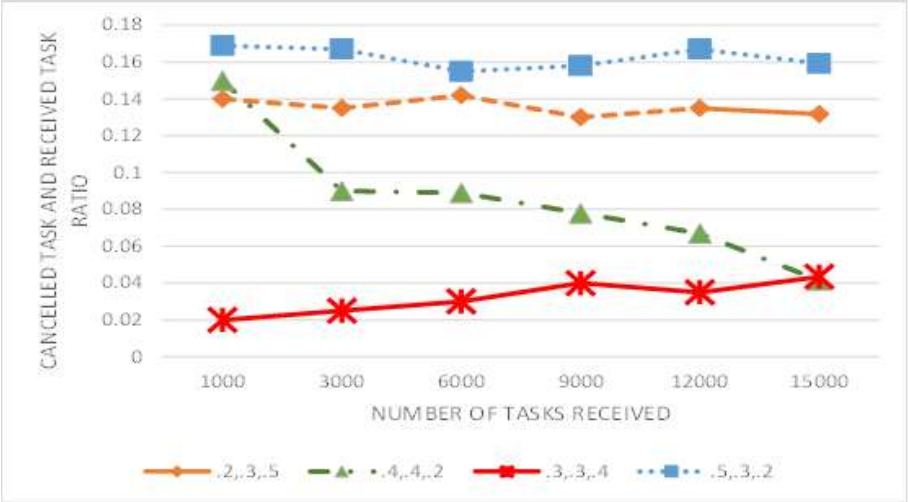


Figure 4.13: Normalised energy consumption w.r.t varying constants

Fig 4.12 , Fig 4.13 and Fig 4.14 depicts that χ=0.4, ψ=0.4, ω=0.2 in Equation4.13 gives the optimised result in terms of time and energy consumption and minimum number of job cancellation.

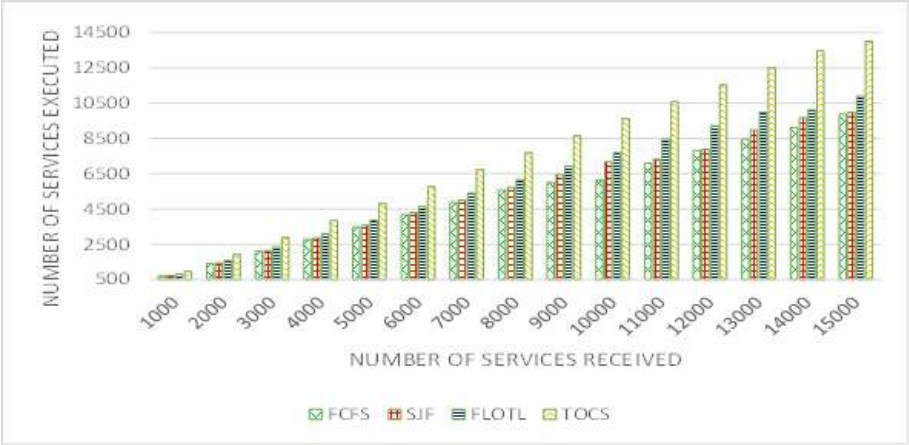


Figure 4.15: Comparison between different strategies w.r.t Number of service received vs Number of service executed

The purpose of this experiment is to evaluate the proposed algorithm against a recent state-of-the art algorithm. In order to achieve this, the proposed algorithm has been compared with Fuzzy Logic for Offloading to the Target Layer (FLOTL) scheme, published in [106]. We have also taken into account the following popular intuitions.

- FCFS(FIRST COME FIRST SERVE): Tasks are assigned to IoT devices in ascending order of ID; tasks that the devices are unable to perform are assigned to the ECU in the same cluster. The task is assigned to an ECU in a separate cluster if the ECU in the same cluster is not available.

- SJF(SHORTEST Job FIRST): It is similar to FCFS with the exception that the tasks are arranged according to CPU cycle requirements.

Fig 4.15 shows the result of the comparison. It is clear from the result that the proposed algorithm is able to achieve better results in terms of reduced number of canceled tasks as the algorithm considers different metrics and three layer architecture for task offloading. From the figure it is found that the proposed algorithm is able to achive 28.1%, 24.29%, and 12.86% less task cancellation than FCFS, SJF, and FLOTL([106])algorithm, respectively.

4.4 Summary

An optimal job offloading algorithm based on the Cuckoo Search Algorithm for the Internet of Things in an edge computing context has been presented in this chapter. The method is able to develop an offloading strategy for applications with data centric, deadline aware jobs in order to optimize service time and resource usage in IoT-Edge environment concurrently reducing the overall energy consumption and percentage of job failure. The technique can handle heterogeneous ECUs with different numbers of virtual machines. The Python programming language is used to implement the intended method in a CISCO packet tracer, and it is demonstrated to function successfully in both even and uneven virtual machine distribution. The outcomes demonstrate how the proposed method outperforms a state of the art technique in terms of fewer job cancellations.

In order to provide an overall energy-efficient job execution strategy in an IoT-edge environment, it is necessary to form an efficient job scheduling algorithm following offloading the jobs in the respective ECUs. Thus, we have concentrated on creating an energy-efficient job scheduling strategy that takes into account job priorities, dependencies, and conflicts in the upcoming chapter.

Journal Communicated

1. **Journal**:

   M.Bakshi,U.Maulik and C.Chowdhury, "Cuckoo Search based Deadline Aware Energy Optimized Task Offloading Strategy," communicated to *SN Computer Science*, Springer.

Chapter 5

Sustainable job scheduling

5.1 Introduction

The key focus of this chapter is to introduce a job scheduling algorithm that considers job priority, inter-job dependencies, job conflicts, and varying ECU capacities. The aim is to enhance the overall performance of ECUs by optimizing resource utilization and energy efficiency. Due to the complexity of the task, the solution space becomes significantly large, prompting the use of meta-heuristic optimization algorithms to navigate it effectively. Among these algorithms, cuckoo search-based optimization seems promising as it facilitates both exploration and exploitation throughout the search process. Therefore, we have developed a Cuckoo Search based Job Scheduling (CSJS) method that addresses the specified conditions and generates an optimized job schedule for edge devices.

5.2 System Model

The architecture consists of three layers.

1. IoT device layer
2. ECU layer
3. Cloud Data Server layer

In IoT device layer, numerous IoT devices are placed. Here, M number of IoT devices are considered to be deployed. Let each IoT device deploys one service that needs to

be processed remotely. That means M number of jobs to be processed, defined as $J = \{j_1, j_2, ..., j_M\}$. Initially, jobs are placed in a waiting queue. Proposed scheduling algorithm would determine which jobs should be placed in which ECU depending upon various criteria so that job execution cost is minimized whereas the ECU utilization would be maximized.

In the ECU layer, there are L number of ECUs deployed where $L << M$, defined as $E = \{E_1, E_2, ..., E_L\}$. Each ECU consists of an access point and a variable number of VMs. Jobs will be placed in the VMs of the respective ECUs according to the scheduling algorithm, such that the energy consumption cost is minimized and simultaneously, the resource utilization is maximized. It is the responsibility of the ECU to execute the jobs or to send them to cloud data server as per the need.

5.2.1 Problem Definition

The problem can be defined as follows.

Let there be M number of jobs and L ECUs. Each ECU has a variable number of VMs. The objective of the algorithm is to produce a job schedule such that

1. Energy consumption is minimized.
2. Resource utilization is maximized.
3. Maintaining job conflict and dependency properties in the schedule.
4. Prioritizing the execution of higher priority, ready for execution jobs.

Some of the constraints are

1. All jobs are non pre-emptive.
2. Any single job will be allocated to a single ECU , though it may require more than one VMs.
3. Any job can start execution only after all the jobs, on which it is dependent has already completed their execution.

4. No two jobs can be placed in the same ECU at the same time which have conflicts among themselves.

To solve the above job scheduling optimization problem, we have proposed a Cuckoo Search based job scheduling algorithm.

5.2.2 Method Overview

All the jobs $\{j_1, j_2, ..., j_M\}$ are placed in the priority queue, with status 0 as defined in equation 5.1, sorted according to their respective priority index $\{p_1, p_2, ..., p_M\}$.

$$s_i = \begin{cases} 0 & \text{if } j_i \text{ is in the waiting queue} \\ 1 & \text{if } j_i \text{ is placed in the ECU} \\ 2 & \text{if } j_i \text{ has been completed and removed from the ECU} \end{cases} \tag{5.1}$$

Initially, all the VMs of all the ECUs $\{E_1, E_2, ..., E_M\}$ are empty. A job j_i is ready to be executed if all the jobs j_k on which it is dependent, as defined in equation 5.2, have been executed.

$$D_i^k = \begin{cases} 1 & \text{if } j_i \text{ is dependent on } j_k \\ 0 & \text{if } j_i \text{ is not dependent on } j_k \end{cases} \tag{5.2}$$

The scheduling algorithm extracts the list of highest priority jobs which are ready to be executed i.e $RD_j = 1$ as per equation 5.3.

$$RD_j = \begin{cases} 1 & \forall \text{ jobs } j_p \text{ if } D_j^p{=}1 \text{ and } s_p{=}2. \\ 0 & \text{Otherwise} \end{cases} \tag{5.3}$$

Then, a set of initial solutions has been generated with all combinations of ready to be executed, highest priority jobs and the ECUs. Now using these initial solutions, a Cuckoo Search based algorithm is executed to select the set of solutions with the highest fitness value. To calculate the highest fitness value, at first, the compatibility of the ECUs with the jobs in the solution is checked. A job is compatible with an ECU if the ECU has sufficient number of available VMs required for the execution of the job and no already

assigned jobs have conflict with the job under consideration, as defined in the equation 5.4.

$$C_i^k = \begin{cases} 1 & \text{if } j_i \text{ has conflict with } j_k \\ 0 & \text{if } j_i \text{ does not have conflict with } j_k \end{cases} \tag{5.4}$$

In the proposed work, in every round, the energy cost and resource utilization have been calculated that are fed in the fitness value calculation so that the optimized job sequence can be selected. The energy cost and the resource utilization calculations are shown in the following subsections.

5.2.2.1 Energy Cost Calculation

For a job that is compatible with an ECU in the solution set, the amount of energy to be consumed, if the job is placed in that ECU, is calculated. The number of active ECUs with active VMs are calculated as follows.

$$AE_{i,r} = \begin{cases} 1 & \text{if } len(JVM_{i,r}) > 0 \\ 0 & \text{Otherwise} \end{cases} \tag{5.5}$$

Equation 5.5 determines whether the i^{th} ECU is active in round r. The i^{th} ECU is considered to be active in round r if there is at least one VM in that ECU, in that particular round, which is executing a job. That means the length of $JVM_{i,r}$ must be greater than 0, where $JVM_{i,r}$ is the list of jobs placed in the i^{th} ECU w.r.t round r.

Whether the j^{th} VM of ECU_i is active in round r is determined as follows.

$$AVM_{i,j,r} = \begin{cases} 1 & \text{if } len(JVM_{i,r}[j]) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.6}$$

$JVM_{i,r}$ is the list of jobs of i^{th} ECU in round r. For example, if the ECU_i has 3 VMs then $JVM_{i,r}$ may contain the list $[[j1], [], [j2]]$. That means in the round r, $VM_{i,1}$ is executing j_1, while $VM_{i,2}$ is not executing any job, and $VM_{i,3}$ is executing j_2. So, $JVM_{i,r}[1] > 0$, $JVM_{i,r}[2] = 0$, and $JVM_{i,r}[3] > 0$. Thus, by equation 5.6, $AVM_{i,1,r} = 1$, $AVM_{i,2,r} = 0$, $AVM_{i,3,r} = 1$. The total energy consumed in round r is calculated using

equation 5.7. The energy is consumed in three ways, through active ECUs (μ per unit), active VMs (ν per unit), and inactive VMs of the active ECUs (Γ per unit). Accordingly, the total energy requirement, TE_r is calculated as follows.

$$TE_r = \mu \times \sum_{p=1}^{i} (AE_{p,r}) + \nu \times \sum_{i=1}^{L} \sum_{j=1}^{VM_i} (AVM_{i,j,r}) + \Gamma \times \sum_{i=1}^{L} AE_i \times \sum_{j=1}^{VM_i} (1 - AVM_{i,j,r}) \quad (5.7)$$

The total energy cost, TE_r is normalized w.r.t the maximum energy consumption per round as follows.

$$NE_r = \frac{TE_r}{(L \times \mu) + \nu \times \sum_{p=1}^{L} VM_p} \quad (5.8)$$

Accordingly, the total energy consumption for T rounds is computed as follows.

$$EC = \sum_{r=1}^{T} TE_r \quad (5.9)$$

5.2.2.2 Resource Utilization

The normalized resource utilization is calculated, assuming the job to be placed in that ECU as follows.

$$NRU_r = \frac{\sum_{i=1}^{L} \sum_{j=1}^{VM_i} AVM_{i,j,r}}{\sum_{i=1}^{L} (AE_{i,r} \times VM_i)} \quad (5.10)$$

It represents ratio of the total number of active VMs with the total number of VMS in all active ECUs. Any active ECU consumes a lot of energy irrespective of how many jobs are placed there. So, the proposed algorithm tries to maximize the utilization of the VMs of the active ECUs before needing to activate an additional ECU. Since the inactive ECUs do not consume any energy therefore, the normalized resource utilization has been calculated taking only the number of active ECUs. The algorithm is detailed in the following section.

5.2.2.3 Algorithm Overview

The proposed Cuckoo Search based Job Scheduling (CSJS) algorithm takes the list of jobs to be executed and returns the optimized job sequence such that the consumption of energy cost is minimized and the resource utilization is maximized.

Since there are M jobs and L ECUs so there will be M^L possible ways of allocation of such M jobs in L ECUs. The jobs considered for execution must have the following properties.

1. The job has not started execution.
2. There exists at least one ECU such that it has sufficient number of empty VMs for execution of the job.
3. All the jobs on which the opted job is dependent have completed their execution.

The suitability of ECU for the execution of the job depends on mainly two factors.

1. There are enough VMs available in the ECU for the execution of the job.
2. No job which has a conflict with the selected job is presently being executed in the ECU.

The CheckCompatibility(job_j, ecu_i) procedure is invoked to check the compatibility between an ECU and a job. It returns 1 if job_j is compatible with ecu_i, otherwise, returns 0. This procedure is invoked by the fitness function F(job_j, ecu_i), while calculating the fitness value of a solution.

Depending on the fitness function, the proposed CSJS algorithm, summarized as Algorithm 4, selects the best suitable solution. For implementation of Cuckoo search, on Steps 9 to 19 of Algorithm 4, the initial nests are considered to be randomly generated sequence of (job,ECU) set. While generating new solutions in step 16, for say cuckoo i, a Levy flight is performed. Here, the consecutive jumps/steps of a cuckoo essentially form a random walk process which obeys a power-law step-length distribution with a heavy tail. Thus, the chances of stucking at a local optima are minimized. As t increases, steps started decreasing and intensification of solution space has been achieved. From here the

algorithm would find out the best Cuckoo, that is the optimized sequence of (job, ECU) set. The Cuckoo Search is executed either up-to a given no of times or till three iterations that the algorithm chooses the best Cuckoo, whichever is less. According to the result produced by the algorithm, the selected jobs are placed to the respective ECUs and the status of the jobs are changed to 2 following equation 5.1. The procedure is repeated till all the VMs are full or there is no ready to be executed job whose status = 0. The jobs start executing. When a job finishes its execution, it is removed from all the VMs which it is acquiring and its status becomes 2. Whenever a VM becomes empty, the procedure of job selection is again repeated, till all the jobs have status= 2. The fitness value is calculated by Algorithm 9 where the normalized resource consumption and resource utilization are taken into account while assigning the jobs to compatible ECUs. The weight for the two factors are determined empirically. It is to be noted that the energy consumption is a cost that should be minimized while resource utilization is the benefit that should be maximized.

```
1  Procedure CheckCompatibility(job_j,ecu_i)
2      R = {NULL};
3      c = 1 if ∀j_k s.tD_j^k == 1, s_k == 2 otherwise 0;
4      if c==1 then
5          for all e_j in ECU_i where (VM_i − length(JVM_{i,r})) − RVM_j >= 0 do
6              if ∄ j_k ∈ e_j suchthat c_i^k == 1 then
7                  return 1;
8              end
9          end
10     end
11     return 0;
```

```
1  Procedure F(job_j,ecu_i)
2      fitval = CheckCompatibility (ecu_i, job_j) × ((Φ×(1−NE_r)+Ψ×NRU_r)/(Φ+Ψ));
3      return fitval;
```

Algorithm 4: CSJS

Input: Job Queue

Output: Optimized job sequence

```
Initially all jobs are placed in a waiting queue J;
J is sorted according to priority ;
r=1 ;
F_j=0 ;
while ∃j_p∈J : s_p! = 2 do
    if (∃E_i∈E : length(JVM_{i,r}) < VM_i)&(∃j_p : RD_p == 1&s_p == 0) then
        Pick the list of jobs j_H ready for execution;
        Generate initial soln (set of nests) with all combinations of jobs
         ∈j_H&E_i∈ECU;
        while (t<MaxGeneration) or (stop criterion) do
            Get a Cuckoo i randomly via Levy flights;
            Evaluate its quality/fitness F_i;
            if (F_i > F_j) then
                Replace j by the new solution;
            end
            Abandon a fraction (pa) of worse nests;
            build new ones at new locations via Levy flights;
            Keep the best solutions (or nests with quality solutions);
            Rank the solutions and find the current best;
        end
        Insert the selected Job j_s to the selected ECU E_q;
        Put s_s = s_s + 1
    else
        Start executing the jobs s.t s_p = 1 i.e; already placed in VMs;
        r=r+1;
        if job j_p is completed then
            remove the job from all VMs which it is acquiring;
            Put s_p = s_p + 1 ;
            continue;
        end
    end
end
```

5.3 Experimental Results and Discussion

In this section, numerical results are presented to evaluate the performance of the proposed CSJS algorithm. In order to validate the results, the proposed algorithm is simulated using the MATLAB simulator [1]. Here, we have assumed number of jobs= 250, and number of ECUs=10. Number of VCs of each ECU arbitrarily varies from 1 to 7. Power consumption by active ECU, active VMs and inactive VMs are μ, ν, Γ respectively following the values stated in [74]. We have also executed the code in Python 10.8.3 version for verification of the opted result.

In Table 5.1, the default parameter settings for evaluating CSJS are listed. The values are taken following the datasheet [74]. A case study is presented first to show the detailed working of the algorithm. For explaining the case study we have taken a small set of 24 jobs, to ease our explanation domain. However, for calculating the results, 250 jobs are considered. A brief discussion of the experimental results is presented in the subsequent subsections.

Table 5.1: Experimental Setup Parameters

KEY PARAMETER	VALUE
M	250
L	10
VMi	[1,7]
μ	300W
ν	50W
Γ	30W

5.3.1 Case Study

A case study has been performed with 25 jobs. In Table 5.2, a job is described by the following parameters.

- Id: Provides the unique identification number of the job.
- Priority: Priority of a particular job.
- Dependency: Provides the list of the job ids, on which the job is dependent.

[1] `https://www.mathworks.com/products/matlab.html`

- Conflict: Provides the list of the job ids with which the particular job has a conflict.
- Duration of a particular job w.r.t round.

Table 5.3 has provided optimized job schedule for the given job description in Table 5.2 as the output of the proposed CSJS algorithm. It has shown round wise details i.e; on which round, which job will be placed on an ECU such that the resource utilization is maximized as well as the energy consumption is minimized. It is to be noted that in Table 5.3 some fields are left blank as no jobs are assigned to the VMs of the ECUs for that particular round. From Table 5.3 it is also seen that any new ECU is assigned only if no other active ECUs have sufficient number of VMs to execute a particular job. The resource utilization is calculated on the basis of the VMs used in the active ECUs. Even with heterogeneous ECUs with job conflicts and dependencies, the resource utilization is found to be around 82% which is quite appreciable.

From Table 5.2 and Table 5.3, it can be observed that the algorithm is focusing on job priority. For example, job 8 and job 22 started their execution first, having priority 1, provided they do not have any job dependency, or job conflict with the jobs in available VMs.

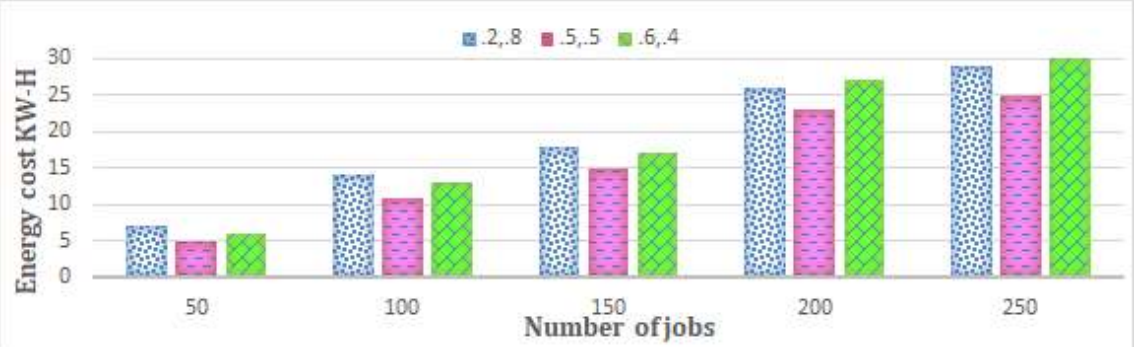
Handling of job dependency by the algorithm is reflected in Table 5.3. For example, job 6 has priority 1, but it is not able to start its execution till round 5 when job 2 finishes its execution, as job 6 is dependent on job 2 (shown in Table 5.2).

The algorithm also handles job conflict as is reflected in Table 5.3. For example, in round 5, job 3 is placed in ECU2, though there was sufficient VM available in ECU1 because job3 has a job conflict with job 8 (shown in Table 5.2).

5.3.2 Performance tuning of the proposed CSJS algorithm

The default values of the setup parameters taken in this section is listed in Table 5.1. Thus, the results are reported for 250 jobs with 10 ECUs. It is important to find out the optimized value of the weight parameters Φ and Ψ used in Algorithm 9 for calculating the optimized job schedule. In Fig. 5.1, the consumed energy cost and resource utilization w.r.t different number of jobs are tested with varying the values of (Φ, Ψ), respectively, where the weight coefficient of each evaluation factor satisfies $\Phi + \Psi = 1$.

(a) Normalized resource utilization with varying constants (Φ, Ψ)



(b) Energy cost KW-H with varying constants

Figure 5.1: Effect on different metrics with varying constants

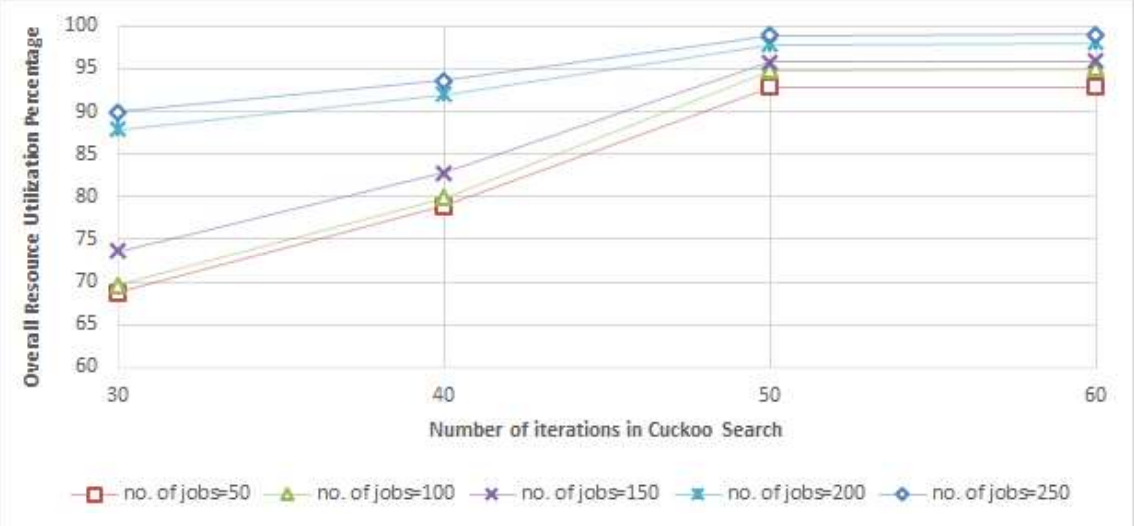


Figure 5.2: Convergence performance of the proposed CSJS algorithm

Table 5.2: Jobs description for performing a case study

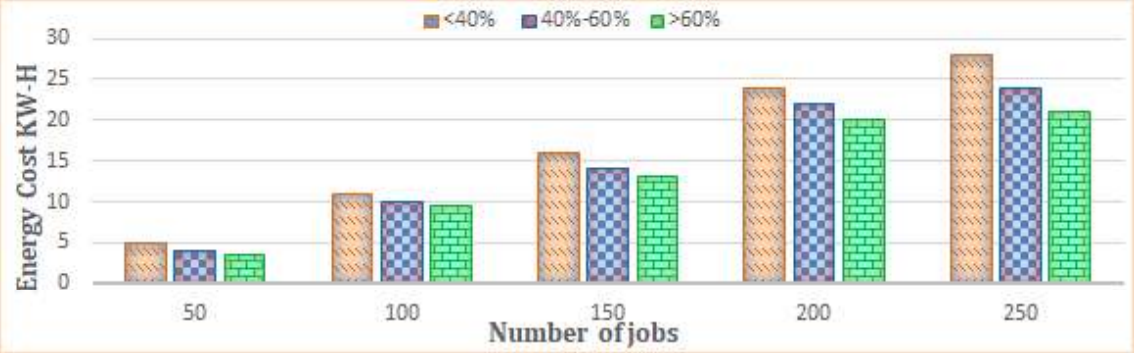
id	Priority	RVM	Dependency	Conflict	Duration
0	6	1	[]	[18, 5]	6
1	3	1	[]	[18, 3]	8
2	5	1	[]	[]	4
3	1	1	[2]	[8]	10
4	2	2	[3, 2, 1]	[3]	6
5	7	1	[]	[17]	10
6	1	1	[2]	[]	13
7	1	1	[2, 6]	[19, 4, 3]	4
8	1	1	[]	[14]	7
9	5	1	[2, 3]	[]	1
10	1	2	[5, 3]	[7, 6]	12
11	5	1	[9]	[]	1
12	4	2	[]	[]	6
13	7	1	[]	[18, 12]	11
14	7	1	[6, 2, 1]	[0, 15]	9
15	2	1	[0]	[14, 3]	4
16	2	1	[]	[2]	5
17	6	2	[]	[]	12
18	3	2	[0, 2]	[]	9
19	2	2	[13, 6, 15]	[]	2
20	2	1	[0, 5]	[19]	5
21	4	2	[18, 12]	[]	6
22	1	2	[]	[]	4
23	5	1	[0, 1]	[7]	9
24	6	2	[11]	[]	6

Different representative value combinations are explored in order to cover the entire horizon. As can be observed from the figures, the value combination ($\Phi = 0.5, \Psi = 0.5$) gives the best results. Thus, equal weightage for both the factors seem to work best for job selection as it maximizes resource utilization (Fig. 5.1a) while consumes less energy as shown in Fig. 5.1b. For rest of the experiments, both the factors are given equal weightage accordingly.

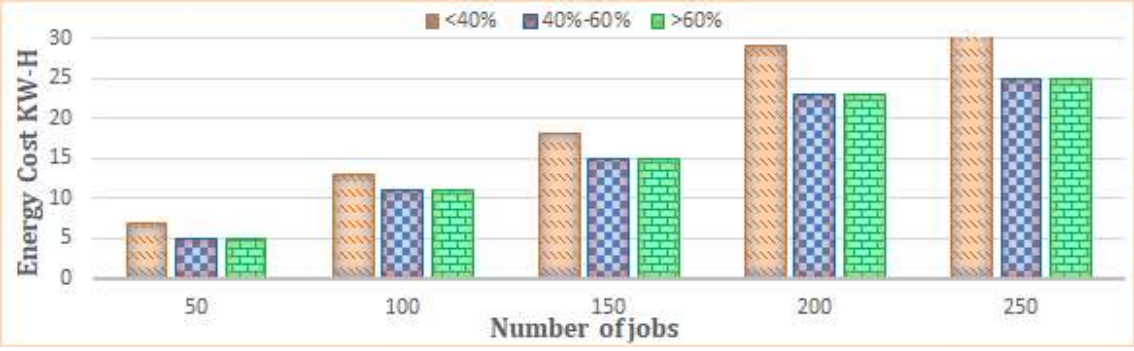
In order to ensure reproducibility of the results it is important to ensure that the proposed algorithm converges after a finite number of iterations. Convergence of the CSJS algorithm is explored and the result is shown in Fig. 5.2. It is found that the outcome stabilizes after 50 iterations for varying number of jobs. Thus, the subsequent results are taken at more than 50 iterations in order to report stable output.

(a) Normalized resource utilization with no job conflict and dependency

(b) Normalized resource utilization with job conflict and dependency

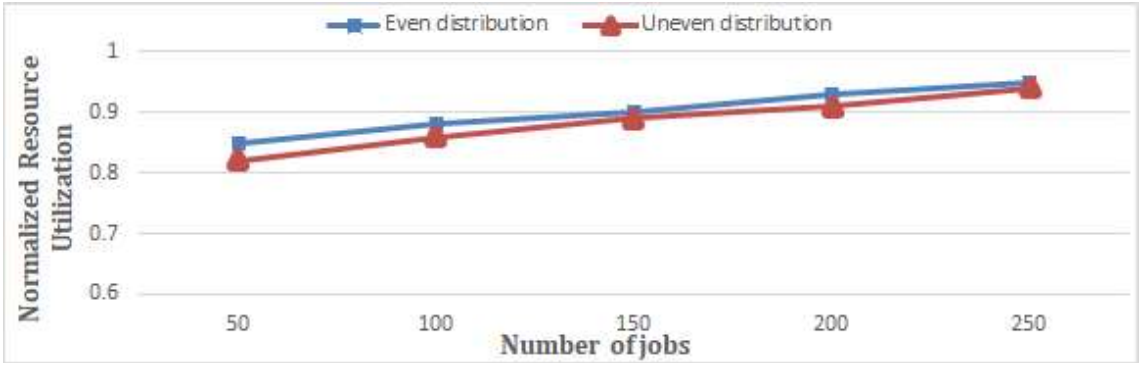


(c) Energy consumption with no job conflict and dependency

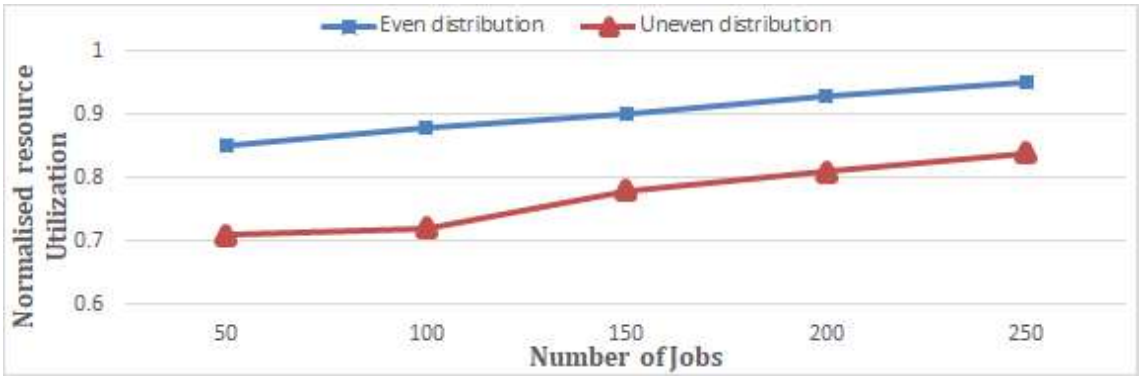


(d) Energy consumption with job conflict and dependency

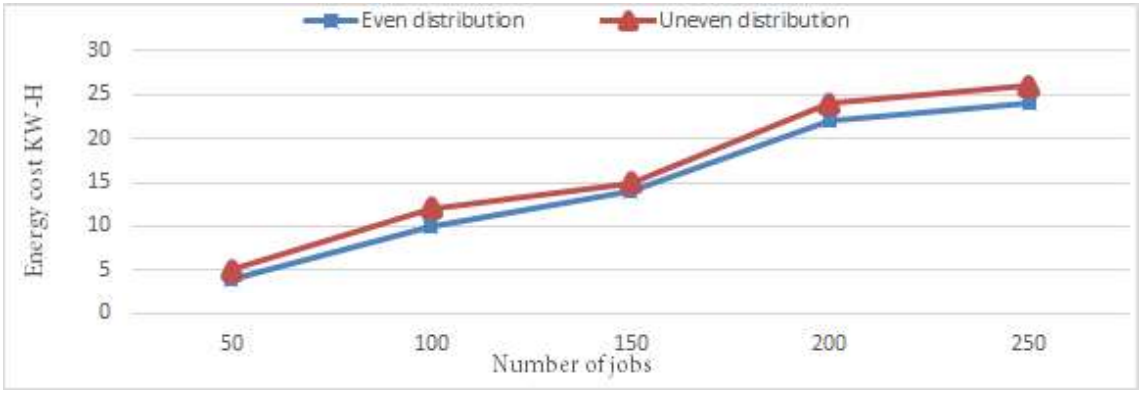
Figure 5.3: Variation of Energy consumption and normalized resource utilization for varying percentage of VMs w.r.t number of jobs



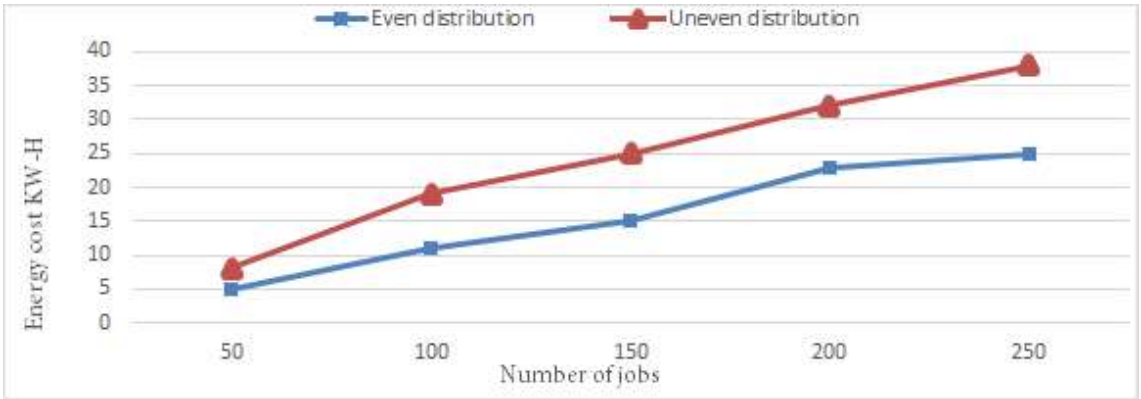
(a) Resource Utilization for homogeneous and heterogeneous ECUs with no job conflict and dependency



(b) Resource Utilization for homogeneous and heterogeneous ECUs with job conflict and dependency



(c) Energy consumption for homogeneous and heterogeneous ECUs with no job conflict and dependency

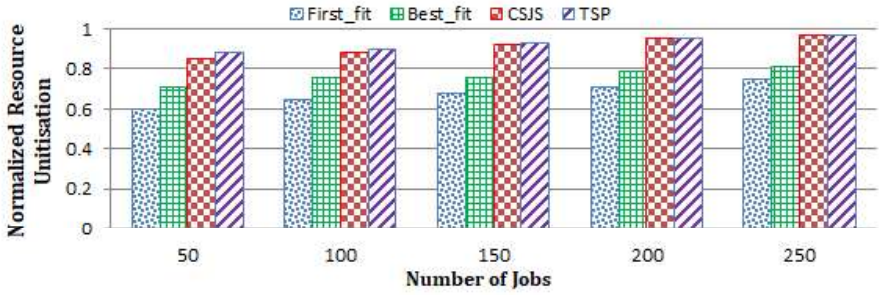


(d) Energy Consumption for homogeneous and heterogeneous ECUs with job conflict and dependency

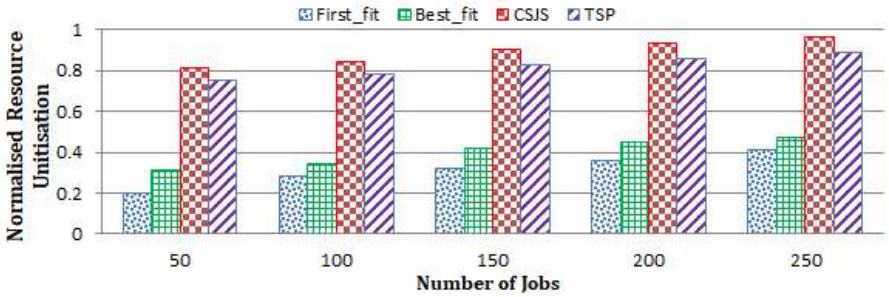
Figure 5.4: Variation of Normalized resource utilization and Energy consumption w.r.t even and uneven distribution of VMs in the ECUs

Table 5.3: Round wise execution details of a case study with jobs described in Table 5.2

Round	ECU1				ECU2		ECU3			ECU4				ECU5		Resource Utilization
1	22	22	8	16	1	2	12	12	0	17	17	13		5		0.816666667
2	22	22	8	16	1	2	12	12	0	17	17	13		5		0.816666667
3	22	22	8	16	1	2	12	12	0	17	17	13		5		0.816666667
4	22	22	8	16	1	2	12	12	0	17	17	13		5		0.816666667
5	6		8	16	1	3	12	12	0	17	17	13		5		0.766666667
6	6		8		1	3	12	12	0	17	17	13		5		0.716666667
7	6	15	8		1	3	18	18		17	17	13		5		0.7
8	6	15			1	3	18	18		17	17	13		5		0.65
9	6	15	23			3	18	18		17	17	13		5		0.6
10	6	15	23			3	18	18		17	17	13		5		0.6
11	6	23	20			3	18	18		17	17	13				0.666666667
12	6	23	20			3	18	18		17	17					0.604166667
13	6	23	20			3	18	18								0.638888889
14	6	23	20			3	18	18								0.638888889
15	6	23	20	9	10	10	18	18		4	4					0.791666667
16	6	23	21	21	10	10	11			4	4					0.708333333
17	6	23	21	21	10	10	24	24		4	4					0.791666667
18	7	14	21	21	10	10	24	24		4	4	19	19			0.916666667
19	7	14	21	21	10	10	24	24		4	4	19	19			0.916666667
20	7	14	21	21	10	10	24	24		4	4					0.791666667
21	7	14	21	21	10	10	24	24								0.888888889
22		14			10	10	24	24								0.638888889
23		14			10	10										0.625
24		14			10	10										0.625
25		14			10	10										0.625
26		14			10	10										0.625



(a) Normalized Resource Utilization without considering job conflicts and dependency



(b) Normalized Resource Utilization considering job conflicts and dependency

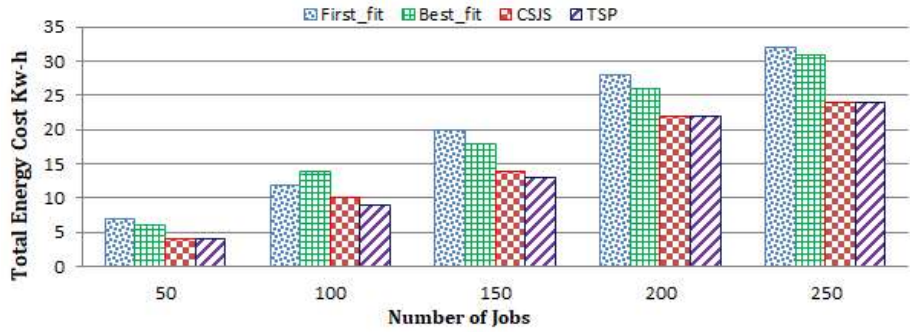
Figure 5.5: Normalized Resource Utilization consumption w.r.t varying number of jobs

5.3.3 Analysis of VM Requirements

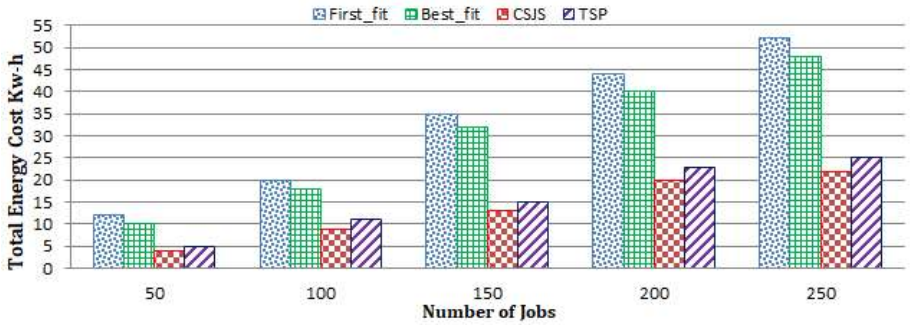
It is important to investigate the required number of VMs based on the demand, that is, the jobs to be assigned for getting the optimized result. An experiment has been conducted, by varying the total number of VMs with respect to different number of jobs. Two different scenarios are considered as follows.

1. Without considering job conflicts and dependency and

2. Considering job conflicts and dependency.



(a) Total energy cost consumption without considering Job conflicts and dependency



(b) Total energy cost consumption considering Job conflicts and dependency

Figure 5.6: Total energy cost consumption w.r.t varying number of jobs

Fig. 5.3 shows the resultant output for the metrics resource utilization (Fig. 5.3a,Fig. 5.3b) and energy cost consumption (Fig. 5.3c,Fig. 5.3d), respectively. It is clearly shown in the figures that when there is no job conflict and dependency, with the increase of number of VMs, the resource utilization increases (Fig. 5.3a) and simultaneously, the consumption of energy is decreased (Fig. 5.3c). However, when job conflict and dependency are considered, the resource utilization gets increased upto 60% (shown in Fig. 5.3b) but beyond that, no further improvement could be observed because of job dependencies. Thus, even if the

edge infrastructure is increased, it cannot be utilized beyond a point since the dependent jobs need to wait till their requisite jobs finish execution.

An experiment is conducted to further dig into the relationship between the ECUs along with its VMs and the number of jobs. Fig. 5.4 shows the results of the experiments that has been conducted with even and uneven distribution of VMs in the ECUs. It is shown in the figure that even distribution, that is, homogeneous VMs give better result in terms of minimum energy consumption and maximum resource utilization. Interestingly, it is also shown that, there is minor difference between the two scenarios when the job conflict and dependency are not considered (Fig. 5.4a, Fig. 5.4c). However, difference in performance could be observed when the job conflict and dependency are considered (Fig. 5.4b, Fig. 5.4d).

5.3.4 Comparison with state-of-the-art techniques

The experiments have been performed to compare the proposed algorithm with the state-of-the-art algorithms. For this purpose, we have compared the proposed algorithm with commonly used strategies and a state-of-the-art scheme TSP (Trust oriented IoT service placement) reported in [74]. The following commonly used intuitions are considered.

- BestFit where the jobs are placed to the ECU which have the least number of available VM, but enough for execution of the job.
- FirstFit where jobs are placed to the first ECU having required number of VMS for execution of jobs.

It is clearly seen from the figures Fig. 5.5(Fig. 5.5a,Fig. 5.5b) and Fig. 5.6(Fig. 5.6a,Fig. 5.6b) that, the proposed algorithm outperforms BestFit and FirstFit methods in terms of maximum resource utilization and minimum energy consumption. The proposed algorithm works in a comparable way with the work reported in [74] when no job conflict and dependency are considered, and performs better than the work in [74] when job conflict and dependency are considered. This is because the proposed algorithm have considered dependency while populating the initial solutions for Cuckoo Search (reflected in line no 3 of Algorithm 4) and have considered job conflict while calculating the fitness value.

5.4 Summary

In this study, we have introduced an optimal job scheduling algorithm for IoT in edge computing environment, based on Cuckoo Search Algorithm. The algorithm is able to produce a suitable job schedule subject to different constraints such as, job priority, dependency among various jobs, and job conflicts. The algorithm ensures maximum resource utilization and minimum energy consumption. The algorithm supports heterogeneous ECUs consisting of varying number of VMs. The designed algorithm is implemented in Matlab and is shown to perform well in different conditions such as, with and without job conflict and dependency, and even and uneven distribution of VMs. The results show how the developed technique works better than a state-of-the-art technique in terms of energy consumption and resource utilization.

List Of Publications

1. **Journal**:

   M.Bakshi, C.Chowdhury and U.Maulik, "Cuckoo search optimization-based energy efficient job scheduling approach for IoT-edge environment," Springer, 2023, DOI: https://doi.org/10.1007/s11227-023-05358-1, 79(16), pp. 18227-18255.

Chapter 6

Conclusion and Future Scope

IoT-Edge-Cloud network paradigm is an emerging paradigm that possesses immense potential for real-life applications. An efficient and feasible job execution strategy in IoT Edge environment is essential to meet the various requirements of the wide range of applications. Heterogeneous deployment environment, varying job requirements, and energy-constraint of IoT nodes are the main challenges tackled in this thesis. Other challenges include limited computation capabilities, uncertainty in job arrivals, deadline sensitive jobs, and mobility of IoT nodes.

This chapter presents the concluding remarks, providing a review of the works documented in this thesis. It includes an analysis of the overall significance of the results and suggests a few directions to extend this work in the future.

6.1 Summary

The work presented in this thesis is summarized as follows:-

- The focus was on three key aspects of job execution in an IoT-Edge-Cloud environment: Edge server placement, job offloading, and task scheduling strategy.
- The study initially involved dividing the area of interest into clusters based on the number of deployed nodes, with the ability to dynamically reconfigure these clusters if nodes become inactive. Optimal positions for deploying Edge server nodes were identified to ensure efficient coverage of the area with minimal energy consumption

and balanced workload distribution.

- Additionally, a job offloading algorithm was developed to determine whether a job should be offloaded based on various quality of service requirements such as deadlines, CPU cycles, and data transmission needs. This algorithm aims to optimize energy usage and prevent missed deadlines.

- Furthermore, a job scheduling approach was proposed to generate an efficient schedule considering job priorities, dependencies, and conflicts in order to optimize resource utilization and energy consumption.

- The thesis ultimately suggests an energy-efficient job execution strategy in an IoT-Edge-Cloud environment, particularly suited for heterogeneous environments with varying VM availability, IoT device numbers, and job QoS requirements.

- The use of swarm intelligence-based algorithms like GSO and Cuckoo Search were employed to address the multiple (sometimes conflicting) optimization objectives posed by dynamic task arrivals and conflicting factors in the environment.

- The performance of the proposed strategies were evaluated through simulations and compared with existing techniques, showing superior outcomes. We have considered a real-life dataset to validate the experimentation. Detailed discussions on the methodologies and results can be found in chapters 3, 4, and 5 of the thesis.

6.2 Summary of Contribution

The chapter-wise contributions are summarized as follows:

Chapter 3 explores a method for choosing cluster heads with Glow worm swarm optimization. The algorithm efficiently splits the network into an ideal number of clusters, maintaining a balance between the edge server's needs and the base stations connected to an edge. The effectiveness of the suggested approach is assessed through thorough experimentations. The algorithm is found to perform remarkably better than existing IoT clustering protocols, even when over 20% of nodes lose power. Over time, the algorithm shows increasing improvements.

Chapter 4 describes a Cuckoo search based task offloading algorithm for IoT networks within an edge computing framework. This algorithm is specifically designed to improve service time and resource utilization for applications with data-centric, deadline-based tasks. A key goal is to minimize energy consumption and task failures in the IoT-Edge environment. Simulations demonstrate that the new algorithm outperforms a state-of-the-art edge computing task offloading strategy. According to experimentations reported here, the work is found to achieve 12.8% fewer task cancellations than a cutting-edge approach.

Chapter 5 presents a task scheduling algorithm utilizing Cuckoo Search for IoT systems in an edge computing environment. This algorithm focuses on three factors namely task priority, dependencies, and conflicts to generate an efficient task schedule. The main goal is to enhance resource allocation and minimize energy consumption. Through simulation, the algorithm proves to outperform existing edge-computing task scheduling methods, particularly when dealing with conflicting or dependent tasks. The study shows a resource utilization rate of over 85% even with the presence of task conflicts and dependencies.

6.3 Future Scope

This thesis explores different approaches to figure out the best way to perform tasks in an IoT-Edge environment. Certain research areas need further exploration. The upcoming section discusses the examination of the thesis's findings that have potential for further expansion.

This study introduces a method for placing energy-efficient edge servers in an IoT setting to distribute workloads effectively and maximize coverage with fewer servers. In reality, edge servers can have varying capacities due to different factors like device types and brands, resulting in differences in computation, storage, and energy consumption. As mobile devices become more intertwined with users, displaying traits like mobility and heterogeneity in performance metrics, future efforts will focus on enhancing edge server placement in IoT by accounting for the diversity, mobility, and social aspects among these servers.

Moreover, if the proposed research work intends to expand to encompass more geographical regions, it will be necessary to ensure that the workload on edge nodes is balanced

effectively to maintain optimal performance. Consequently, it will be essential to develop suitable solutions to address the load balancing issues on edge nodes in the future.

In the task-based edge computing model, the research has mostly considered simulating a limited number of high priority tasks generated by the load generator, whereas in reality, the task quantities are not fixed. This disparity may lead to additional communication overhead between IoT-edge-cloud in real-time, which will require further investigation.

The task offloading strategy outlined in this thesis does not currently take into account the dependencies and dependability of various tasks being offloaded. This aspect will be a focus of our future research.

The thesis provides a task scheduling mechanism that assumes tasks are non-preemptive, but in reality, there are both preemptive and non-preemptive tasks that need to be managed. This diversity makes scheduling more challenging, requiring research into new mechanisms to handle hybrid tasks and resources. Additionally, task migration may be necessary in real-life situations, which need to be considered in the future.

Performance metrics determination for edge server placement in a flood-prone area

To assess the effectiveness of edge server placement in areas prone to flooding, several key metrics should be taken into account:

- Latency: Minimizing latency is essential for quick response times, particularly during flood situations where immediate data processing and rapid decision-making are vital.
- Load Balancing Efficiency: Effective load balancing is crucial to manage high traffic loads during flooding events, preventing any single server from becoming overloaded.
- Data Processing Speed: Enhanced data processing speed enables faster decision-making, which is vital in emergencies.
- Energy Consumption: In flood-affected regions, energy-efficient servers are important to maintain longer operation times during potential power outages.
- Geographical Coverage: Proper server placement guarantees that all devices in flood-prone regions have access to low-latency data processing.

- Edge Server Accessibility: Ensuring that devices can consistently access edge computing resources without interruptions, even under challenging conditions.
- Environmental Durability: The server hardware must be robust enough to withstand adverse environmental factors, such as high humidity and possible water contact.

Utilizing these metrics allows for a comprehensive evaluation of edge server placement strategies, making certain that they effectively fulfill the requirements for real-time data processing and communication during flood scenarios.

Bibliography

[1] Roberto Verdone, Davide Dardari, Gianluca Mazzini, and Andrea Conti. *Wireless sensor and actuator networks: technologies, analysis and design.* Academic Press, 2010.

[2] Lina Xu, Gregory O'Hare, and Rem Collier. A smart and balanced energy-efficient multihop clustering algorithm (smart-beem) for mimo iot systems in future networks. *Sensors*, 17(7):1574, 2017.

[3] X. Sun and N. Ansari. Dynamic resource caching in the iot application layer for smart cities. *IEEE Internet of Things Journal*, 5(2):606–613, 2018.

[4] V Subramaniyaswamy, Gunasekaran Manogaran, R Logesh, V Vijayakumar, Naveen Chilamkurti, D Malathi, and N Senthilselvan. An ontology-driven personalized food recommendation in iot-based healthcare system. *The Journal of Supercomputing*, 75(6):3184–3216, 2019.

[5] Brian D Noble, Mahadev Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R Walker. Agile application-aware adaptation for mobility. *ACM SIGOPS Operating Systems Review*, 31(5):276–287, 1997.

[6] Shaxun Chen. Content delivery network. *Cina: Springer-Verlag*, 1998.

[7] Jason Flinn and Mahadev Satyanarayanan. Energy-aware adaptation for mobile applications. *ACM SIGOPS Operating Systems Review*, 33(5):48–63, 1999.

[8] Mahadev Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal communications*, 8(4):10–17, 2001.

[9] John Dilley, Bruce Maggs, Jay Parikh, Harald Prokop, Ramesh Sitaraman, and Bill Weihl. Globally distributed content delivery. *IEEE Internet Computing*, 6(5):50–58, 2002.

[10] Jayashree Ravi, Weisong Shi, and Cheng-Zhong Xu. Personalized email management at network edges. *IEEE Internet Computing*, 9(2):54–60, 2005.

[11] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4):14–23, 2009.

[12] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16, 2012.

[13] Jackson He, Li Jiang, and Jerry Xie. China: A booming market for cloud computing. *Intel® Technology Journal*, 16(4), 2012.

[14] Xing Chen and Guizhong Liu. Energy-efficient task offloading and resource allocation via deep reinforcement learning for augmented reality in mobile edge networks. *IEEE Internet of Things Journal*, 8(13):10843–10856, 2021.

[15] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing—a key technology towards 5g. *ETSI white paper*, 11(11):1–16, 2015.

[16] Fabio Giust, Xavier Costa-Perez, and Alex Reznik. Multi-access edge computing: An overview of etsi mec isg. *IEEE 5G Tech Focus*, 1(4):4, 2017.

[17] Volkan Gezer, Jumyung Um, and Martin Ruskowski. An extensible edge computing architecture: Definition, requirements and enablers. *Proceedings of the UBICOMM*, 2017.

[18] H Grigoryan, S Shoukourian, Gurgen Harutyunyan, Yervant Zorian, and Costas Argyrides. Advanced ecc-based fit rate mitigation technique for automotive socs. In *2018 IEEE International Test Conference (ITC)*, pages 1–6. IEEE, 2018.

[19] Weisong Shi, George Pallis, and Zhiwei Xu. Edge computing [scanning the issue]. *Proceedings of the IEEE*, 107(8):1474–1481, 2019.

[20] En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. Edge AI: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications*, 19(1):447–457, 2019.

[21] Yunlong Lu, Xiaohong Huang, Ke Zhang, Sabita Maharjan, and Yan Zhang. Communication-efficient federated learning for digital twin edge networks in industrial iot. *IEEE Transactions on Industrial Informatics*, 17(8):5709–5718, 2020.

[22] Salam Hamdan, Moussa Ayyash, and Sufyan Almajali. Edge-computing architectures for internet of things applications: A survey. *Sensors*, 20(22):6441, 2020.

[23] Shichao Guan and Azzedine Boukerche. Intelligent edge-based service provisioning using smart cloudlets, fog and mobile edges. *IEEE Network*, 36(2):139–145, 2022.

[24] Koustabh Dolui and Soumya Kanti Datta. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6. IEEE, 2017.

[25] Abdulsalam Yassine, M Shamim Hossain, Ghulam Muhammad, and Mohsen Guizani. Cloudlet-based intelligent auctioning agents for truthful autonomous electric vehicles energy crowdsourcing. *IEEE Transactions on Vehicular Technology*, 69(5):5457–5466, 2020.

[26] Yannis Nikoloudakis, Spyridon Panagiotakis, Evangelos Markakis, Evangelos Pallis, George Mastorakis, Constantinos X Mavromoustakis, and Ciprian Dobre. A fog-based emergency system for smart enhanced living environments. *IEEE Cloud Computing*, 3(6):54–62, 2016.

[27] Rashmi Shrivastava and Manju Pandey. Real time fall detection in fog computing scenario. *Cluster Computing*, 23(4):2861–2870, 2020.

[28] Mikel Celaya-Echarri, Iván Froiz-Míguez, Leyre Azpilicueta, Paula Fraga-Lamas, Peio Lopez-Iturri, Francisco Falcone, and Tiago M Fernández-Caramés. Building

decentralized fog computing-based smart parking systems: from deterministic propagation modeling to practical deployment. *IEEE Access*, 8:117666–117688, 2020.

[29] Yong Xiao and Marwan Krunz. Adaptivefog: A modelling and optimization framework for fog computing in intelligent transportation systems. *IEEE Transactions on Mobile Computing*, 21(12):4187–4200, 2021.

[30] Abdelhamied A Ateya, Ammar Muthanna, Andrey Koucheryavy, Yassine Maleh, and Ahmed A Abd El-Latif. Energy efficient offloading scheme for mec-based augmented reality system. *Cluster Computing*, 26(1):789–806, 2023.

[31] Haoji Hu, Hangguan Shan, Chuankun Wang, Tengxu Sun, Xiaojian Zhen, Kunpeng Yang, Lu Yu, Zhaoyang Zhang, and Tony QS Quek. Video surveillance on mobile edge networks—a reinforcement-learning-based approach. *IEEE Internet of Things Journal*, 7(6):4746–4760, 2020.

[32] Raghubir Singh and Sukhpal Singh Gill. Edge AI: a survey. *Internet of Things and Cyber-Physical Systems*, 3:71–92, 2023.

[33] Ali Hassan Sodhro, Sandeep Pirbhulal, and Victor Hugo C De Albuquerque. Artificial intelligence-driven mechanism for edge computing-based industrial applications. *IEEE Transactions on Industrial Informatics*, 15(7):4235–4243, 2019.

[34] Daniel Situnayake and Jenny Plunkett. *AI at the Edge*. " O'Reilly Media, Inc.", 2023.

[35] Amit Kumar, Tahrat Tazrin, Arti Sharma, Shivani Chaskar, Sadman Sakib, Mostafa M Fouda, and Zubair Md Fadlullah. Ai-aided secured ecg live edge monitoring system with a practical use-case. In *Secure Edge Computing*, pages 177–197. CRC Press, 2021.

[36] Mehdi Nazari Cheraghlou, Ahmad Khadem-Zadeh, and Majid Haghparast. Eft: Novel fault tolerant management framework for wireless sensor networks. *Wireless Personal Communications*, pages 1–19, 2019.

[37] Xiang Sun and Nirwan Ansari. Traffic load balancing among brokers at the IoT application layer. *IEEE Transactions on Network and Service Management*, 15(1):489–502, 2017.

[38] Gary White, Vivek Nallur, and Siobhán Clarke. Quality of service approaches in iot: A systematic mapping. *Journal of Systems and Software*, 132:186–203, 2017.

[39] J.Sathish Kumar and Mukesh A. Zaveri. Clustering Approaches for Pragmatic Two-Layer IoT Architecture. *Wireless Communications and Mobile Computing*, 2018:16, 2018.

[40] Yoonyoung Sung, Sookyoung Lee, and Meejeong Lee. A multi-hop clustering mechanism for scalable iot networks. *Sensors*, 18(4):961, 2018.

[41] Nafaâ Jabeur, Ansar Ul-Haque Yasar, Elhadi Shakshuki, and Hedi Haddad. Toward a bio-inspired adaptive spatial clustering approach for iot applications. *Future Generation Computer Systems*, 2017.

[42] Daewon Lee and HwaMin Lee. IoT service classification and clustering for integration of IoT service platforms. *The Journal of Supercomputing*, 74(12):6859–6875, 2018.

[43] VA Geetha, Pranesh V Kallapur, and Sushma Tellajeera. Clustering in wireless sensor networks: Performance comparison of leach & leach-c protocols using ns2. *Procedia Technology*, 4:163–170, 2012.

[44] Ping Ding, JoAnne Holliday, and Aslihan Celik. Distributed energy-efficient hierarchical clustering for wireless sensor networks. In *International conference on distributed computing in sensor systems*, pages 322–339. Springer, 2005.

[45] Xuxun Liu. A survey on clustering routing protocols in wireless sensor networks. *sensors*, 12(8):11113–11153, 2012.

[46] Ir Riri Fitri Sari. Bioinspired algorithms for internet of things network. In *2017 4th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*, pages 1–1. IEEE, 2017.

[47] Zhihua Cui, Yang Cao, Xingjuan Cai, Jianghui Cai, and Jinjun Chen. Optimal leach protocol with modified bat algorithm for big data sensing systems in internet of things. *Journal of Parallel and Distributed Computing*, 132:217–229, 2019.

[48] Anuj Sehgal, Vladislav Perelman, Siarhei Kuryla, and Jurgen Schonwalder. Management of resource constrained devices in the internet of things. *IEEE Communications Magazine*, 50(12):144–149, 2012.

[49] Charith Perera, Prem Prakash Jayaraman, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Mosden: An internet of things middleware for resource constrained mobile devices. In *2014 47th Hawaii International Conference on System Sciences*, pages 1053–1062. IEEE, 2014.

[50] Shangguang Wang, Yali Zhao, Jinlinag Xu, Jie Yuan, and Ching-Hsien Hsu. Edge server placement in mobile edge computing. *Journal of Parallel and Distributed Computing*, 127:160–168, 2019.

[51] Jingrong Wang, Kaiyang Liu, and Jianping Pan. Online uav-mounted edge server dispatching for mobile-to-mobile edge computing. *IEEE Internet of Things Journal*, 7(2):1375–1386, 2019.

[52] Yang Zhang, Dusit Niyato, and Ping Wang. Offloading in mobile cloudlet systems with intermittent connectivity. *IEEE Transactions on Mobile Computing*, 14(12):2516–2529, 2015.

[53] Yuanzhe Li, Ao Zhou, Xiao Ma, and Shangguang Wang. Profit-aware edge server placement. *IEEE Internet of Things Journal*, 9(1):55–67, 2021.

[54] Yuanzhe Li and Shangguang Wang. An energy-aware edge server placement algorithm in mobile edge computing. In *2018 IEEE International Conference on Edge Computing (EDGE)*, pages 66–73. IEEE, 2018.

[55] Guangming Cui, Qiang He, Feifei Chen, Hai Jin, and Yun Yang. Trading off between user coverage and network robustness for edge server placement. *IEEE Transactions on Cloud Computing*, 2020.

[56] Feng Zeng, Yongzheng Ren, Xiaoheng Deng, and Wenjia Li. Cost-effective edge server placement in wireless metropolitan area networks. *Sensors*, 19(1):32, 2018.

[57] Shahrukh Khan Kasi, Mumraiz Khan Kasi, Kamran Ali, Mohsin Raza, Hifza Afzal, Aboubaker Lasebae, Bushra Naeem, Saif Ul Islam, and Joel JPC Rodrigues. Heuristic edge server placement in industrial internet of things and cellular networks. *IEEE Internet of Things Journal*, 8(13):10308–10317, 2020.

[58] Manal Alqarni, Asma Cherif, and Entisar Alkayyal. ODM-BCSA: An offloading decision-making framework based on binary cuckoo search algorithm for mobile edge computing. *Computer Networks*, 226:109647, 2023.

[59] Wenhao Fan, Li Gao, Yi Su, Fan Wu, and Yuan'an Liu. Joint dnn partition and resource allocation for task offloading in edge-cloud-assisted IoT environments. *IEEE Internet of Things Journal*, 2023.

[60] Zhongmin Wang, Yurong Ding, Xiaomin Jin, Yanping Chen, and Cong Gao. Task offloading for edge computing in industrial internet with joint data compression and security protection. *The Journal of Supercomputing*, 79(4):4291–4317, 2023.

[61] Ting Wang, Yuxiang Deng, Zhao Yang, Yang Wang, and Haibin Cai. Parameterized deep reinforcement learning with hybrid action space for edge task offloading. *IEEE Internet of Things Journal*, 2023.

[62] Hao Zhao, Jiahui Xu, Pei Li, Wei Feng, Xin Xu, and Yingbiao Yao. Energy minimization partial task offloading with joint dynamic voltage scaling and transmission power control in fog computing. *IEEE Internet of Things Journal*, 2023.

[63] Ihsan Ullah, Hyun-Kyo Lim, Yeong-Jun Seok, and Youn-Hee Han. Optimizing task offloading and resource allocation in edge-cloud networks: a DRL approach. *Journal of Cloud Computing*, 12(1):112, 2023.

[64] Hansong Xu, Jun Wu, Qianqian Pan, Xing Liu, and Christos Verikoukis. Digital Twin and Meta RL Empowered Fast-Adaptation of Joint User Scheduling and Task Offloading for Mobile Industrial IoT. *IEEE Journal on Selected Areas in Communications*, 2023.

[65] Mohammad Yahya Akhlaqi and Zurina Binti Mohd Hanapi. Task offloading paradigm in mobile edge computing-current issues, adopted approaches, and future directions. *Journal of Network and Computer Applications*, 212:103568, 2023.

[66] Posham Bhargava Reddy and Chapram Sudhakar. An osmotic approach-based dynamic deadline-aware task offloading in edge–fog–cloud computing environment. *The Journal of Supercomputing*, pages 1–23, 2023.

[67] Abegaz Mohammed Seid, Gordon Owusu Boateng, Bruce Mareri, Guolin Sun, and Wei Jiang. Multi-agent drl for task offloading and resource allocation in multi-uav enabled IoT edge network. *IEEE Transactions on Network and Service Management*, 18(4):4531–4547, 2021.

[68] Ziru Zhang, Nianfu Wang, Huaming Wu, Chaogang Tang, and Ruidong Li. Mr-dro: A fast and efficient task offloading algorithm in heterogeneous edge/cloud computing environments. *IEEE Internet of Things Journal*, 2021.

[69] Jiaying Meng, Haisheng Tan, Xiang-Yang Li, Zhenhua Han, and Bojie Li. Online deadline-aware task dispatching and scheduling in edge computing. *IEEE Transactions on Parallel and Distributed Systems*, 31(6):1270–1286, 2019.

[70] Lingfeng Cai, Xianglin Wei, Changyou Xing, Xia Zou, Guomin Zhang, and Xiulei Wang. Failure-resilient dag task scheduling in edge computing. *Computer Networks*, page 108361, 2021.

[71] Jin Wang, Yang Liu, Shan Ren, Chuang Wang, and Wenbo Wang. Evolutionary game based real-time scheduling for energy-efficient distributed and flexible job shop. *Journal of Cleaner Production*, 293:126093, 2021.

[72] Mingyan Li, Cailian Chen, Huaqing Wu, Xinping Guan, and Xuemin Shen. Age-of-information aware scheduling for edge-assisted industrial wireless networks. *IEEE Transactions on Industrial Informatics*, 17(8):5562–5571, 2020.

[73] Ihsan Ullah and Hee Yong Youn. Task classification and scheduling based on k-means clustering for edge computing. *Wireless Personal Communications*, 113(4):2611–2624, 2020.

[74] Xiaolong Xu, Xihua Liu, Zhanyang Xu, Fei Dai, Xuyun Zhang, and Lianyong Qi. Trust-oriented iot service placement for smart cities in edge computing. *IEEE Internet of Things Journal*, 7(5):4084–4091, 2020.

[75] Atakan Aral and Tolga Ovatman. A decentralized replica placement algorithm for edge computing. *IEEE transactions on network and service management*, 15(2):516–529, 2018.

[76] Duong Tuan Nguyen, Chuan Pham, Kim Khoa Nguyen, and Mohamed Cheriet. Placement and chaining for run-time iot service deployment in edge-cloud. *IEEE Transactions on Network and Service Management*, 17(1):459–472, 2019.

[77] Irfan Mohiuddin and Ahmad Almogren. Workload aware vm consolidation method in edge/cloud computing for IoT applications. *Journal of Parallel and Distributed Computing*, 123:204–214, 2019.

[78] Ashkan Yousefpour, Ashish Patil, Genya Ishigaki, Inwoong Kim, Xi Wang, Hakki C Cankaya, Qiong Zhang, Weisheng Xie, and Jason P Jue. Fogplan: a lightweight qos-aware dynamic fog service provisioning framework. *IEEE Internet of Things Journal*, 6(3):5080–5096, 2019.

[79] BV Natesha and Ram Mohana Reddy Guddeti. Adopting elitism-based genetic algorithm for minimizing multi-objective problems of iot service placement in fog computing environment. *Journal of Network and Computer Applications*, 178:102972, 2021.

[80] Tao Ouyang, Zhi Zhou, and Xu Chen. Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing. *IEEE Journal on Selected Areas in Communications*, 36(10):2333–2345, 2018.

[81] Balázs Sonkoly, Dávid Haja, Balázs Németh, Márk Szalay, János Czentye, Róbert Szabó, Rehmat Ullah, Byung-Seo Kim, and László Toka. Scalable edge cloud platforms for IoT services. *Journal of Network and Computer Applications*, 170:102785, 2020.

[82] Jonghwa Choi and Sanghyun Ahn. Scalable service placement in the fog computing environment for the iot-based smart city. *Journal of Information Processing Systems*, 15(2):440–448, 2019.

[83] Arkadiusz Madej, Nan Wang, Nikolaos Athanasopoulos, Rajiv Ranjan, and Blesson Varghese. Priority-based fair scheduling in edge computing. In *2020 IEEE 4th International Conference on Fog and Edge Computing (ICFEC)*, pages 39–48. IEEE, 2020.

[84] Jin Wang, Yiquan Cao, Bin Li, Hye-jin Kim, and Sungyoung Lee. Particle swarm optimization based clustering algorithm with mobile sink for wsns. *Future Generation Computer Systems*, 76:452–457, 2017.

[85] L Yong H Mufang Z Ke X Renrong Y Xiuwu, L Qin. Uneven clustering routing algorithm based on glowworm swarm optimization. *Ad Hoc Networks*, 93:101923, 2019.

[86] Seyed Mostafa Bozorgi, Mahdi Rohani Hajiabadi, Ali Asghar Rahmani Hosseinabadi, and Arun Kumar Sangaiah. Clustering based on whale optimization algorithm for IoT over wireless nodes. *Soft Computing*, 25(7):5663–5682, 2021.

[87] Walid Osamy, Ahmed A El-Sawy, and Ahmed Salim. Csoca: Chicken swarm optimization based clustering algorithm for wireless sensor networks. *IEEE Access*, 8:60676–60688, 2020.

[88] Daniel Puschmann, Payam Barnaghi, and Rahim Tafazolli. Adaptive clustering for dynamic iot data streams. *IEEE Internet of Things Journal*, 4(1):64–74, 2016.

[89] Lingjuan Lyu, Jiong Jin, Sutharshan Rajasegarar, Xuanli He, and Marimuthu Palaniswami. Fog-empowered anomaly detection in iot using hyperellipsoidal clustering. *IEEE Internet of Things Journal*, 4(5):1174–1184, 2017.

[90] Sameh Ben Fredj, Mathieu Boussard, Daniel Kofman, and Ludovic Noirie. A scalable iot service search based on clustering and aggregation. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pages 403–410. IEEE, 2013.

[91] Qiang Fan and Nirwan Ansari. Application aware workload allocation for edge computing-based iot. *IEEE Internet of Things Journal*, 5(3):2146–2153, 2018.

[92] Ola Salman, Imad Elhajj, Ali Chehab, and Ayman Kayssi. Iot survey: An sdn and fog computing perspective. *Computer Networks*, 143:221–246, 2018.

[93] Nadeem Javaid, Saman Cheema, Mariam Akbar, Nabil Alrajeh, Mohamad Souheil Alabed, and Nadra Guizani. Balanced energy consumption based adaptive routing for iot enabling underwater wsns. *IEEE Access*, 5:10040–10051, 2017.

[94] Shi Yan, Mugen Peng, and Xueyan Cao. A game theory approach for joint access selection and resource allocation in uav assisted iot communication networks. *IEEE Internet of Things Journal*, 6(2):1663–1674, 2018.

[95] Gopika Premsankar, Bissan Ghaddar, Mario Di Francesco, and Rudi Verago. Efficient placement of edge computing devices for vehicular applications in smart cities. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2018.

[96] Futian Wang, Xingxiang Huang, Hongfang Nian, Qiang He, Yun Yang, and Cheng Zhang. Cost-effective edge server placement in edge computing. In *Proceedings of the 2019 5th international conference on systems, control and Communications*, pages 6–10, 2019.

[97] Yan Guo, Shangguang Wang, Ao Zhou, Jinliang Xu, Jie Yuan, and Ching-Hsien Hsu. User allocation-aware edge cloud placement in mobile edge computing. *Software: Practice and Experience*, 50(5):489–502, 2020.

[98] Kun Cao, Liying Li, Yangguang Cui, Tongquan Wei, and Shiyan Hu. Exploring placement of heterogeneous edge servers for response time minimization in mobile edge-cloud computing. *IEEE Transactions on Industrial Informatics*, 17(1):494–503, 2020.

[99] Xingcun Li, Feng Zeng, Guanyun Fang, Yinan Huang, and Xunlin Tao. Load balancing edge server placement method with QoS requirements in wireless metropolitan area networks. *IET Communications*, 14(21):3907–3916, 2020.

[100] Feiyan Guo, Bing Tang, Linyao Kang, and Li Zhang. Mobile edge server placement based on bionic swarm intelligent optimization algorithm. In *Collaborative Computing: Networking, Applications and Worksharing: 16th EAI International Conference, CollaborateCom 2020, Shanghai, China, October 16–18, 2020, Proceedings, Part II 16*, pages 95–111. Springer, 2021.

[101] Yuanyi Chen, Yihao Lin, Zengwei Zheng, Peng Yu, Jiaxing Shen, and Minyi Guo. Preference-aware edge server placement in the internet of things. *IEEE Internet of Things Journal*, 9(2):1289–1299, 2021.

[102] Yanling Shao, Zhen Shen, Siliang Gong, Hanyao Huang, et al. Cost-aware placement optimization of edge servers for IoT services in wireless metropolitan area networks. *Wireless Communications and Mobile Computing*, 2022, 2022.

[103] Ali Asghari, Hossein Azgomi, Ali Abbas Zoraghchian, and Abbas Barzegarinezhad. Energy-aware server placement in mobile edge computing using trees social relations optimization algorithm. *The Journal of Supercomputing*, 80(5):6382–6410, 2024.

[104] Om-Kolsoom Shahryari, Hossein Pedram, Vahid Khajehvand, and Mehdi Dehghan TakhtFooladi. Energy-efficient and delay-guaranteed computation offloading for fog-based IoT networks. *Computer Networks*, 182:107511, 2020.

[105] Huaming Wu, Katinka Wolter, Pengfei Jiao, Yingjun Deng, Yubin Zhao, and Minxian Xu. Eedto: An energy-efficient dynamic task offloading algorithm for blockchain-enabled IoT-edge-cloud orchestrated computing. *IEEE Internet of Things Journal*, 8(4):2163–2176, 2020.

[106] Jaber Almutairi and Mohammad Aldossary. A novel approach for IoT tasks offloading in edge-cloud environments. *Journal of Cloud Computing*, 10(1):1–19, 2021.

[107] Xueshuo Chen, Yuxing Mao, Hongyu Wang, Yihang Xu, Danyang Li, Siyang Liu, and Xianping Zhao. Data-driven task offloading method for resource-constrained terminals via unified resource model. *IEEE Internet of Things Journal*, 2023.

[108] Wenquan Jin, Sunhwan Lim, Sungpil Woo, Chanwon Park, and Dohyeun Kim. Decision-making of iot device operation based on intelligent-task offloading for im-

proving environmental optimization. *Complex & Intelligent Systems*, 8(5):3847–3866, 2022.

[109] Han Xiao, Changqiao Xu, Yunxiao Ma, Shujie Yang, Lujie Zhong, and Gabriel-Miro Muntean. Edge intelligence: A computational task offloading scheme for dependent IoT application. *IEEE Transactions on Wireless Communications*, 21(9):7222–7237, 2022.

[110] Jin Wang, Wenbing Wu, Zhuofan Liao, Arun Kumar Sangaiah, and R Simon Sherratt. An energy-efficient off-loading scheme for low latency in collaborative edge computing. *IEEE Access*, 7:149182–149190, 2019.

[111] Hongjian Li, Peng Zheng, Tiantian Wang, Jingjing Wang, and Tongming Liu. A multi-objective task offloading based on BBO algorithm under deadline constrain in mobile edge computing. *Cluster Computing*, 26(6):4051–4067, 2023.

[112] Shuyue Ma, Shudian Song, Lingyu Yang, Jingmei Zhao, Feng Yang, and Linbo Zhai. Dependent tasks offloading based on particle swarm optimization algorithm in multi-access edge computing. *Applied Soft Computing*, 112:107790, 2021.

[113] Maryam Keshavarznejad, Mohammad Hossein Rezvani, and Sepideh Adabi. Delay-aware optimization of energy consumption for task offloading in fog environments using metaheuristic algorithms. *Cluster Computing*, pages 1–29, 2021.

[114] Nebojsa Bacanin, Milos Antonijevic, Timea Bezdan, Miodrag Zivkovic, K Venkatachalam, and Sharaf Malebary. Energy efficient offloading mechanism using particle swarm optimization in 5g enabled edge nodes. *Cluster Computing*, 26(1):587–598, 2023.

[115] Kaixin Li, Jie Zhao, Jintao Hu, and Ying Chen. Dynamic energy efficient task offloading and resource allocation for noma-enabled iot in smart buildings and environment. *Building and Environment*, 226:109513, 2022.

[116] Jiwei Huang, Songyuan Li, and Ying Chen. Revenue-optimal task scheduling and resource management for IoT batch jobs in mobile edge computing. *Peer-to-Peer Networking and Applications*, 13(5):1776–1787, 2020.

[117] Guisheng Fan, Liang Chen, Huiqun Yu, and Wei Qi. Multi-objective optimization of container-based microservice scheduling in edge computing. *Computer Science and Information Systems*, (00):41–41, 2020.

[118] Kaihua Jiang, Hong Ni, Peng Sun, and Rui Han. An improved binary grey wolf optimizer for dependent task scheduling in edge computing. In *2019 21st International Conference on Advanced Communication Technology (ICACT)*, pages 182–186. IEEE, 2019.

[119] Liuyan Liu, Haisheng Tan, Shaofeng H-C Jiang, Zhenhua Han, Xiang-Yang Li, and Hong Huang. Dependent task placement and scheduling with function configuration in edge computing. In *2019 IEEE/ACM 27th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2019.

[120] Olena Skarlat, Matteo Nardelli, Stefan Schulte, Michael Borkowski, and Philipp Leitner. Optimized IoT service placement in the fog. *Service Oriented Computing and Applications*, 11(4):427–443, 2017.

[121] Rakib Hossen, Md Whaiduzzaman, Mohammed Nasir Uddin, Md Islam, Nuruzzaman Faruqui, Alistair Barros, Mehdi Sookhak, Md Mahi, Julkar Nayeen, et al. Bdps: An efficient spark-based big data processing scheme for cloud fog-iot orchestration. *Information*, 12(12):517, 2021.

[122] Amina Mseddi, Wael Jaafar, Halima Elbiaze, and Wessam Ajib. Joint container placement and task provisioning in dynamic fog computing. *IEEE Internet of Things Journal*, 6(6):10028–10040, 2019.

[123] P Sharma. Energy efficient target set selection and buffer management for d2d mobile data offloading. *International Journal of Data and Network Science*, 5(1):1–10, 2021.

[124] Shashank Swarup, Elhadi M Shakshuki, and Ansar Yasar. Energy efficient task scheduling in fog environment using deep reinforcement learning approach. *Procedia Computer Science*, 191:65–75, 2021.

[125] Sadoon Azizi, Mohammad Shojafar, Jemal Abawajy, and Rajkumar Buyya. Deadline-aware and energy-efficient iot task scheduling in fog computing systems: A semi-greedy approach. *Journal of Network and Computer Applications*, 201:103333, 2022.

[126] Mustafa Ibrahim Khaleel. Efficient job scheduling paradigm based on hybrid sparrow search algorithm and differential evolution optimization for heterogeneous cloud computing platforms. *Internet of Things*, page 100697, 2023.

[127] M Santhosh Kumar and Ganesh Reddy Karri. Eeoa: Cost and energy efficient task scheduling in a cloud-fog framework. *Sensors*, 23(5):2445, 2023.

[128] KN Krishnanand and Debasish Ghose. Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. *Swarm intelligence*, 3(2):87–124, 2009.

[129] Andras Varga. A practical introduction to the OMNeT++ simulation framework. In *Recent Advances in Network Simulation*, pages 3–51. Springer, 2019.

[130] Asanga Udugama, Anna Förster, Jens Dede, and Vishnupriya Kuppusamy. Simulating opportunistic networks with omnet++. In *Recent Advances in Network Simulation*, pages 425–449. Springer, 2019.

[131] Oladiran G Olaleye, Alaa Ali, Dmitri Perkins, and Magdy Bayoumi. Modeling and performance simulation of PULSE and MCMAC protocols in RFID-based IoT network using OMNeT++. In *2018 IEEE International Conference on RFID (RFID)*, pages 1–5. IEEE, 2018.

[132] Alfonso Ariza and Vincenzo Inzillo. Inetmanet framework. In *Recent Advances in Network Simulation*, pages 107–138. Springer, 2019.

[133] Xin-She Yang and Suash Deb. Engineering optimisation by cuckoo search. *International Journal of Mathematical Modelling and Numerical Optimisation*, 1(4):330–343, 2010.

[134] Tarun Kumar Ghosh, Sanjoy Das, Subhabrata Barman, and Rajmohan Goswami. A comparison between genetic algorithm and cuckoo search algorithm to minimize the makespan for grid job scheduling. In Sudip Kumar Sahana and Sujan Kumar Saha, editors, *Advances in Computational Intelligence*, pages 141–147, Singapore, 2017. Springer Singapore.

[135] Xin-She Yang, Mehmet Karamanoglu, T. O. Ting, and Yu-Xin Zhao. Applications and analysis of bio-inspired eagle strategy for engineering optimization. 25(2), 2014.

[136] Xin-She Yang and Mehmet Karamanoglu. Swarm intelligence and bio-inspired computation: an overview. *Swarm intelligence and bio-inspired computation*, pages 3–23, 2013.

[137] Takuro Inoue, Ailixier Aikebaier, Tomoya Enokido, and Makoto Takizawa. Power consumption and processing models of servers in computation and storage based applications. *Mathematical and Computer Modelling*, 58(5-6):1475–1488, 2013.

[138] Mehmet Güçyetmez and Husham Sakeen Farhan. Enhancing smart grids with a new iot and cloud-based smart meter to predict the energy consumption with time series. *Alexandria Engineering Journal*, 79:44–55, 2023.

[139] Sahar Adabi Ali Rezaee. Jobs (DAG workflow) and tasks dataset with near 50k job instances and 1.3 millions of tasks. `https://zenodo.org/records/4667690`.

The Journal of Supercomputing
https://doi.org/10.1007/s11227-020-03536-z

Energy-efficient cluster head selection algorithm for IoT using modified glow-worm swarm optimization

Mohana Bakshi[1] · **Chandreyee Chowdhury**[2] · **Ujjwal Maulik**[2]

Accepted: 23 November 2020

Abstract

Internet of Things (IoT) is a new age technology that connects virtually every IP-enabled things in the world. Because of the constrained nature of the resources, energy-efficient clustering plays a very important role in successful implementation of these networks and also in increasing its lifetime. The proposed methodology provides an adaptive cluster head selection algorithm based on glow-worm swarm optimization algorithm. Most of the existing approaches groups nodes into clusters containing a fixed number of nodes. This is not applicable in certain cases, such as, when many of the nodes are dead. In the proposed approach, the number of nodes in each cluster is not fixed, and it automatically changes according to the number of alive nodes in the network, which increases the lifetime of the network. The proposed approach also ensures the minimum overlapping of the clusters by selecting geographically distributed cluster heads which increases the energy efficiency of the network by minimising the communication overhead. Repeated selection of cluster head also helps in load balancing in the network. We have implemented the proposed algorithm in OMNET++ simulator and demonstrated how it behaves with different densities of sensor deployment and communication ranges. The proposed algorithm is found to work notably well as compared to state-of-the-art IoT clustering protocol even when more than 20% nodes run out of energy. As time progresses, the improvement is even more apt.

Keywords Internet of Things · Clustering · Glow-worm swarm optimization · Swarm optimization · Energy efficiency

✉ Chandreyee Chowdhury
chandreyee.chowdhury@gmail.com

Mohana Bakshi
mohanaprofile@gmail.com

Ujjwal Maulik
ujjwal.maulik@jadavpuruniversity.in

[1] Kendriya Vidyalaya AFS Bagdogra, Jadavpur University, Kolkata, India

[2] Jadavpur University, Kolkata, India

Published online: 02 January 2021

1 Introduction

Internet of Things (IoT) [1] refers to the level of network interconnectivity between IP-enabled physical beings or non-physical entities. It then enables these things to collect and exchange data. Things, in the context of IoT, refers to a variety of devices with built-in sensors enabling them to transfer useful data to the cloud to be processed and saved [2].

IoT is becoming increasingly popular and is involved in many real-life applications such as smart city, smart agriculture, smart environment monitoring, smart water, security monitoring, healthcare systems, military services, and industrial control [3, 4].

For providing these various ranges of applications, the IoT devices placed over a particular area may collect and transmit the sensed or their context data to the servers over the Internet in a periodic manner or in real time. The size of data may vary from the small ones, such as temperature or humidity, to the relatively larger ones, such as images [5]. For a large-scale IoT network, having each IoT device connected to the server with an individual connection may result in the excessive requirement of Internet connections, as well as the waste of wireless resources of an IoT network. Clustering is one of the most important things in IoT [6]. An effective topology is necessary so that the real-world things can optimize their communication, can conserve limited energy, and be able to increase the connectivity between neighbouring peers and offer better services. This is due to the fact that IoT is actually an opportunistic network and at the same time all real-world things are not IP-enabled [7]. To meet these goals, the research and development communities have accepted clustering as an efficient approach for its well-known advantages.

In this regard, clustering provides energy-efficient routing by minimizing the number of participating nodes in the route formation. It also allows data aggregation and provides Quality of Service (QoS) [7]. Clustering for multi-layer IoT architecture is also reported in the literature where a few IP-enabled nodes in one layer connect to the cluster heads of the sensing devices in the lower layer [8]. Moreover, clustering also improves the network scalability and optimizes lifetime of a network for large-scale deployments. Current research efforts on IoT clustering (e.g. [2, 3, 9–11]) are benefiting from the state-of-the art research for wireless sensor networks (WSNs), particularly since the sensors are the basic components of IoT and WSNs. To achieve numerous benefits of clustering algorithms, state-of-the-art technologies have been found to chose the cluster head selection approach. As the cluster heads are expected to have high processing power and can bear more communication load, many methods are introduced to elect the cluster head in an optimized way [12–14].

Different bioinspired algorithms are also used for clustering which are modelled on the basis of living systems [10, 15, 16]. Particle swarm optimization (PSO), artificial bee colony approach have been used in many techniques for this purpose. Few other works have proposed firefly-inspired algorithms where the “real-world things” carry out intense competitions to be cluster heads (CHs),

while trying to attract peers to their clusters. As sensors imitate a major part in the growing domain of Internet of Things (IoT), these algorithms are expected to perform well in the dynamic field while simultaneously adapting to contextual changes and optimizing use of the limited resources [17, 18].

The motivation behind the work is that most of the existing clustering algorithms divide the network into a predetermined number of clusters, which is not suitable when many of the nodes in the network are dead. The main contribution of the proposed work is self-adaptation by rearranging the clusters when most of the cluster heads are dead along with choosing the cluster heads in such a way that energy conservation by the network is maximized.

In the proposed architecture, the devices of the IoT network are grouped into clusters and a representative device for each cluster is selected, known as cluster head (CH), that collects the data from the rest of the devices in each cluster and communicates with the server on behalf of the other devices instead of having a per-device Internet connection and communication. Conservation of the Internet connectivity and the wireless network resources could be made possible by the proposed architecture.

The main two contributions of this paper are:

- The paper presents a clustering algorithm using modified GSO (glow-worm swarm optimization) which is able to divide the IoT network into optimised number of clusters, such that the communication overhead is minimum and the lifetime of the network is optimised.
- The proposed methodology repeats the cluster-head selection, so that the load is uniformly distributed among the nodes. Again, the geographic distribution of the CHs severely influences the overall energy consumption of the network. For prolonging the lifetime of the network, the proposed algorithm ensures that the cluster heads are spread evenly.

The rest of the paper is organized as follows: Sect. 2 presents the literature survey followed by a brief description of the proposed methodology in Sect. 3. Section 4 summarizes the experimental setup and analysis of the experimental results, while Sect. 5 concludes.

2 State-of-the-art clustering protocols

In this section, the existing clustering protocols applicable in the field of IoT and WSN are discussed briefly. Several works are proposed in the literature to enable the advantageous features of clustering in the above-mentioned networks (Table 1).

In MHCM [9], authors have proposed a clustering algorithm which minimizes the number of Internet connections with optimum delay. Their proposed mechanism is comprised of two steps: computing the smallest set of coordinators which covers all IoT nodes in the network N within a maximum hop count constraint H. This is done by repeatedly selecting a coordinator node $\in N$ which can reach the largest number of member nodes within H in a greedy manner. This is followed by optimizing a

Table 1 Comparison between existing clustering algorithms

Algorithm	Energy	Reliability	Latency	QoE awareness	Benchmark
EPMS, 2017 [19]	✓	X	X	X	LEACH, PEGASIS
ASFiCA, 2017 [10]	✓	✓	✓	X	HRPM, GMR
MHCM, 2018 [9]	✓	X	✓	X	LEACH, HEED
SBEEM, 2018 [2]	✓	X	X	✓	BEEM
UCRA-GSO, 2019 [20]	✓	X	X	X	LEACH, EMR, USC

total count of the selected coordinators during the first step by rearranging the mapping of member nodes to the coordinator.

In SBEEM [2], authors have proposed a balanced energy-efficient clustering algorithm, named as BEE. It can elect CHs according to both energy consumption and sensor distributions. It not only extends the networks longevity, but also maintains the network coverage. The authors assumed that all the IoT devices are capable of MIMO. If a node is selected as CH, it turns on MIMO to receive data from different sources through different communication channels. Then, the CHs compress the received data and transmit the data back to the BS. A sensing node only selects one communication interface and a CH for transmission through the Smart-BEEM algorithm.

Swarm intelligence algorithms are also proposed for cluster head selection of IoT. In ASFiCA [10], authors have proposed a firefly-based clustering algorithm. In the micro-clustering phase of ASFiCA, real-world things compete among themselves and self-organize for forming a cluster. After this, in the macro-clustering phase, again the clusters compete to integrate with the small neighbouring clusters. The above approach is implemented for IoT clusters for self-adaptation and selecting/rejecting things depending on their performance in the network and its current deployment area.

In EPMS [19], authors have proposed a particle swarm optimization-based clustering algorithm with mobile sink for wireless sensor network. For a typical sensor network with N' sensors, the network is divided into M' clusters. First, the network region partition line is determined by using the PSO algorithm. In the cluster head selection phase, according to the coordinates of the nodes in each region, the centre of gravity of the region is calculated. Then, the distance from the node to the centre of gravity is obtained. Then, the average residual energy of all nodes in each cluster is calculated. The selected cluster head node has the highest residual energy. A Hello packet is broadcasted by the mobile sink node within two hop range from the cluster head. The mobile sink node selects the cluster with maximum average remaining energy. The collected data by the cluster head are transmitted to the mobile sink node. After a certain duration, the Hello packet is broadcasted again and the sink node moves to the next position.

In UCRA-GSO [20], authors have proposed a GSO-based clustering routing algorithm. For clustering, in the fitness function of GSO, the local density of each cluster head, the average distance within cluster, the energy consumption of nodes within a

cluster and the dispersibility of the cluster head have been taken into account. Initially, a node having residual energy more than the average residual energy is considered as a cluster head; then, the GSO algorithm has been used to determine the optimal clustering method. After clustering a spanning tree has been formed among the selected cluster heads. If the distance between the cluster head and the base station is less than the communication range, then the cluster head directly transfers the data; otherwise, adjacent cluster head with the lowest cost function that is closer to the base station is selected as the next hop.

Most existing clustering algorithms focus on maintaining the number of sensors that are still alive to extend the longevity, ignoring the distribution of the sensors [21–23]. The coverage of a network is highly determined by the sensor distribution, and it is crucial in most systems, like in smart city, smart agriculture, smart environment monitoring, healthcare systems, etc. [24–27]. Our proposed methodology takes into account that some nodes may be dead, after certain round, therefore, rearranges the clusters to extend the lifetime of the network. Moreover, it ensures that the cluster heads are equally distributed in the region of interest.

3 Proposed methodology

The concept of GSO is applied here to solve the dynamic cluster formation and cluster head selection problem of IoT. The GSO mechanism is modified to suit for the problem. Each IoT node is considered to be a glow worm, that is, a candidate solution. The proposed GSO-based cluster-head selection scheme is divided into three phases: sensor luciferin update phase, cluster formation phase, and neighbourhood range update phase. The overview of GSO is presented first. The proposed algorithm and its complexity analysis are presented in subsequent subsections.

3.1 Overview of GSO

The GSO algorithm was first introduced in [28]. The agents in the GSO algorithm, called glow-worms, carry a luminescence quantity called luciferin. Each glow-worm is attracted by the brighter glow of other neighbouring glow-worms. Based on this local decision range, eventually a global solution is reached. In GSO, a swarm is composed of N agents called glow-worms. A state of a glow-worm i at time t can be described by the following set of variables: a position in the search space ($x^i(t)$)), a luciferin level ($l^i(t)$)) and a neighbourhood range ($r^i(t)$)). GSO algorithm describes how these variables change over time.

Initially, agents are randomly distributed in the search space. Other parameters are initialized by predefined constants. Each next iteration is composed of three phases: luciferin-level update, glow-worm movement, and neighbourhood range update.

To encode the fitness of the current position of a glow-worm i in the luciferin level, the following formula is used:

$$l^i(t) = (1-\rho)l^i(t-1) + \gamma^J((x^i(t)) \tag{1}$$

where ρ is the luciferin decay constant, γ is the luciferin enhancement constant, and J is an objective function.

Then, each glow-worm tries to find its neighbours. In GSO, a glow-worm j is a neighbour of a glow-worm i only if the distance between glow-worms i and j is shorter than the neighbourhood range ($r^i(t)$)), and additionally, glow-worm j has to shine brighter than $i(l^j(t) > l^i(t))$. If one glow-worm has multiple neighbours, it chooses one at random with probability proportional to the luciferin level of this neighbour. Finally, a glow-worm moves one step in direction of the chosen neighbour. Step size is constant and equals s.

In the last phase, the neighbourhood range $r_i(t)$ is updated in order to limit the range of the communication from an ensemble of agents. The following formula is used:

$$r^i(t+1) = \min\left\{r_\mathrm{s}, \max\left[0, r^i(t) + \beta\left(n_\mathrm{d} - \left|n^i(t)\right|\right)\right]\right\} \tag{2}$$

where r_s is the sensor range (a constant, which limits the size of the neighbourhood range), n_d is the desired number of neighbours, $\left|n^i(t)\right|$ is the number of neighbours of a glow-worm i at time t, and β is a model constant.

3.2 System model

Given a set of sensors, $S := \left\{s_1, s_2, ...s_{\mathrm{NoS}}\right\}$, with the following properties.

- Sensing range of each node: r_s.
- Decision range of each node: r_d. Initially $r_\mathrm{d} = r_\mathrm{s}$.
- Initial voting index of each sensor : $v_0 = 0$.
- Initial energy level of each sensor $= e_0$.
- Initial luciferin intensity of each sensor: l_0.
- Let CH $:= \left\{c_1, c_2, ...c_k\right\}$ denote the set of cluster heads and $N := \left\{n_1, n_2, ...n_k\right\}$ denote the number of members in each cluster. Initially, each node is treated as cluster head, i.e. every node is forming a cluster with only one member, that is, $n_i = 1 \forall i$.
- Desired number of nodes in each cluster $:= M$, where M is dependent on the sensing radius of the deployed nodes.
- All the nodes have their unique id.

As a response of Hello message for neighbour discovery, nodes would send their unique id. The sender node is able to compute the distance depending upon the response time taken by the receiver node. There is no need to know the position of all the nodes.

After deployment of the nodes, the sink node gathers the initial properties of each node. After that, it initiates the cluster head selection algorithm for the first time. The selected cluster head information is sent to the nodes after successful execution

of the algorithm. Thereafter, whenever energy of any cluster head becomes less than a threshold value (insufficient residual energy to perform as a cluster head), it sends a request to the sink node for re-invoking the algorithm. The proposed modified GSO cluster head selection algorithm contains three phases:

- Luciferin Update Phase
- Cluster Formation Phase
- Neighbourhood-Range Updation Phase

Different phases of the algorithm are described in the following subsections. The proposed modified GSO algorithm for cluster head selection is summarized as Algorithm 1.

3.2.1 Luciferin update phase

Initially, each sensor has its own luciferin intensity. The luciferin intensity of each sensor node in the modified GSO algorithm is updated depending upon its

- Present luciferin intensity $l(t)$.
- Remaining residual energy $e(t)$. The more the residual energy, more will be the luciferin value as the cluster heads consume more energy than other nodes.
- Number of neighbour nodes within its decision range $|H_t|$. The higher the connectivity index, higher will be its luciferin value as that reduces the number of required clusters.
- Voting index $v(t)$. Higher voting index is proportional to its luciferin index as it will help the selection of cluster heads geographically distributed and hence will improve the coverage of the network.

Thus, the sensor luciferin update rule for sensor s_i is given by

$$s_i.l(t+1) = \frac{w \times s_i.l(t) + x \times \frac{s_i.e(t)}{e_0} + y \times \frac{s_i.|H_t|}{M} + z \times \frac{s_i.v(t)}{\text{NoS}}}{w + x + y + z} \tag{3}$$

where w, x, y, z are constants denoting weights of the factors.

Since according to luciferin index a node is voted in the voting phase, it is important to update the luciferin value in a significant way. In the above equation, four main factors have been considered.

Algorithm 1: Modified GSO for clustering

Struct *Node* **contains**
- l(t) := Luciferin value of node at time t;
- N(id,d):= Set of all one hop neighbour id with distance d;
- r_s:= Sensing Range;
- r_d(t):= Decision Range at time t;
- e(t):= Energy level at time t;
- v(t):=voting index at time t;

Input: node structure details of every node at t=0
Output: selected cluster head for each node

$S := \{s_1, s_2, ...s_{NoS}\}$ *the set of sensor nodes to be clustered*;
iter_max := *maximum number of iterations*;
M := *desired number of neighbour nodes*;
$t := 0$;
H_t:=N;
$CH = \Phi$;
while $(t \leq iter\_max)$ **do**
  /* Luciferin update phase */
  for *each sensor* s_i **do**
    $s_i.l\,(t+1) = \frac{w \times s_i.l(t) + x \times \frac{s_i.e(t)}{e_0} + y \times \frac{s_i.|H_t|}{M} + z \times \frac{s_i.v(t)}{NoS}}{w+x+y+z}$;
  end
  /* Cluster Formation phase */
  for *each sensor* s_i **do**
    $s_i.H_t(id, d) = find\_neighbours(s_i.r_d(t))$;
    $k = find\_max\_luciferin\_node(s_i.H_t)$;
    $s_k.v(t+1) = s_k.v(t) + 1$;
  end
  for *each sensor* $s_i \not\subseteq CH$ **do**
    $k = find\_max\_voted\_node(s_i.H\,(t))$;
    $s_i.ch = s_k$;
    $CH = CH \bigcup s_k$;
    $s_k.ch = s_k$;
  end
  if *No change in cluster head selection for three consecutive iterations* **then**
    exit from loop ;
  /* Decision range update phase */
  for *each sensor* s_i **do**
    $s_i.r_d(t+1) = \min\{s_i.r_s, max\{0, s_i.r_d(t) + \alpha\,(M - |s_i.H_t|)\}\}$;
  end
  t=t+1;
end
for *each sensor* s_i **do**
  $send\_cluster\_head(s_i, s_i.ch)$;
  $send\_member\_node(s_i.ch, s_i)$;
end

3.2.2 Cluster formation phase

In the modified GSO algorithm, instead of glow-worm movement phase, a new phase named cluster formation phase is introduced. This phase is further divided into two subphases (1) voting phase and (2) cluster formation phase as described below.

- *Voting Phase* In the find_neighbours() procedure as described in Algorithm 1, the "Hello" messages are sent to all single hop neighbours. Each node that receives a "Hello" message, sends a reply to its sender along with its unique id.

  During the cluster head selection, a voting mechanism is executed to select the cluster head with respect to every sensor node. On behalf of every node, a vote is registered for the node which has the highest luciferin value among its neighbours. In case of a tie, that is, if more than one nodes have same and highest luciferin value, then the vote is registered for the node with lowest index.
- *Cluster Head Selection Phase* All the non-cluster-head node n becomes the member of the cluster with cluster head c_i which has the maximum voting index among node n's neighbours, and the selected cluster head node becomes its own cluster head.

Thus, in the voting phase, each node votes the node with the highest luciferin value in its neighbourhood, and voting index of each node is calculated. In the cluster formation phase, all the voted nodes are informed that they have been selected as cluster heads and also the set of nodes which have voted it become the member nodes of its respective cluster. This voting phase also prevents the overlapping of the clusters.

3.2.3 Neighbourhood-range updation phase

During the neighbourhood range update phase, the neighbourhood range of each sensor s_i is updated based on the following rule:

$$s_i.r_{\mathrm{d}}(t+1) = \min\left\{s_i.r_{\mathrm{s}}, \max\left\{0, s_i.r_{\mathrm{d}}(t) + \alpha\left(M - \left|s_i.H_t\right|\right)\right\}\right\} \tag{4}$$

where α is a constant parameter.

The above procedure is repeated until all the clusters have the desired number of neighbours or the clusters have become stable, that is, no changes have been found in cluster head selection phase for 3 consecutive iterations.

At the end of the execution of the algorithm, the cluster head selection information for each node is sent to them. Also, the nodes which are selected as the cluster head are notified by the sink node.

The entire procedure is summarized as Algorithm 1.

3.3 Complexity analysis

Control message overhead is an essential metric for protocol complexity analysis. In this section, we have analyzed the complexity of the proposed algorithm.

Lemma 1 *The control message complexity of the algorithm is $O(N)$.*

Proof Different types of message passing used in the proposed algorithm are as follows:

Hello_Message: From each node to their H one-hop neighbour.

Reply_Message: From each node to their P sender nodes of Hello_Message.
Details_Message: From each node to sink node.
Cluster_Head_Selection_Message: From sink node to each node

Number of one-hop neighbour of each node H is trivially less than or equal to N, where N is the number of deployed nodes. Therefore, $O(H) = O(N)$.

Number of sender nodes P of hello_message is also trivially less than or equal to N. Therefore, $O(P) = O(N)$.

$\therefore$ $O(H) + O(P) + O(N) + O(N)$, that is, $O(N) + O(N) + O(N) + O(N)$, that implies, $O(N)$. □

4 Experimental results

The experiments are conducted using OMNET++ network simulator to validate the proposed clustering technique. The experimental setup is described first followed by a brief discussion of the results.

4.1 OMNET++

It is an extensible, modular, and component-based, object-oriented C++ simulation library and framework used for network simulation such as wired and wireless communication networks and queueing network [29]. It provides graphical user interface with eclipse-based IDE and a host of other tools (OMNET++) [30]. OMNET++ is more scalable than other simulators for large-scale IoT simulation as reported by [31].

INET framework [32] contains models for Internet stack, wired, and wireless link layer protocols that makes it suitable for IoT applications.

4.2 Results and discussion

As a proof of concept, a INET-based application is developed to test the performance of the proposed algorithm.

Nodes are deployed randomly in the experimental region. Figure 1 shows a sample deployment of the network in INET platform having 50 nodes where the sink is located at the centre of the region. The nodes with rectangular boxes are selected as cluster heads by the proposed modified GSO algorithm. It can be observed from the figure that the cluster heads are equally distributed throughout the region of interest.

Let us consider the case as shown in Fig. 2, say N1, N2, N3, N4, N5 are neighbours to each other. Suppose in the voting phase it is seen that N2 and N5 have same luciferin value and both of them are highest among other neighbours. Then according to the algorithm if there is a tie between N2 and N5, then vote will be registered for N2 as it has minimum identity number.

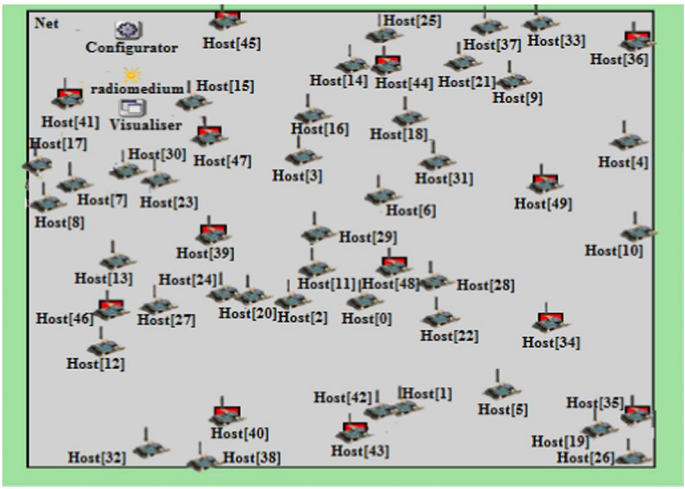


Fig. 1 Deployment of the network in INET

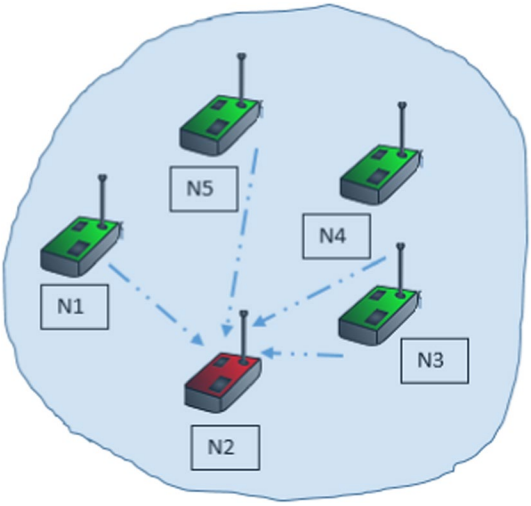


Fig. 2 Cluster head selection in case of a tie

Consider the following scenario as shown in Fig. 3. N1, N2 and N3 are neighbours of N4, and these three nodes vote N4. Node N5 has neighbour nodes N4, N6 and N7. These three nodes vote N5. According to the algorithm, since N4 is already chosen as a cluster head, it will not be any node member of any other cluster. So N1, N2, N3, N4 will form a cluster and N5, N6 and N7 will form another cluster.

After clusters are formed, each node sends data at a constant rate of 4 packets per second. Each packet can contain at most 8 kbits, i.e. 32 kbits/s. Thus, in each round of simulation, each node/the network communicates 32 kbits in 1 s. The default communication range for the nodes is 70–100 m following IEEE 802.15.4 standard.

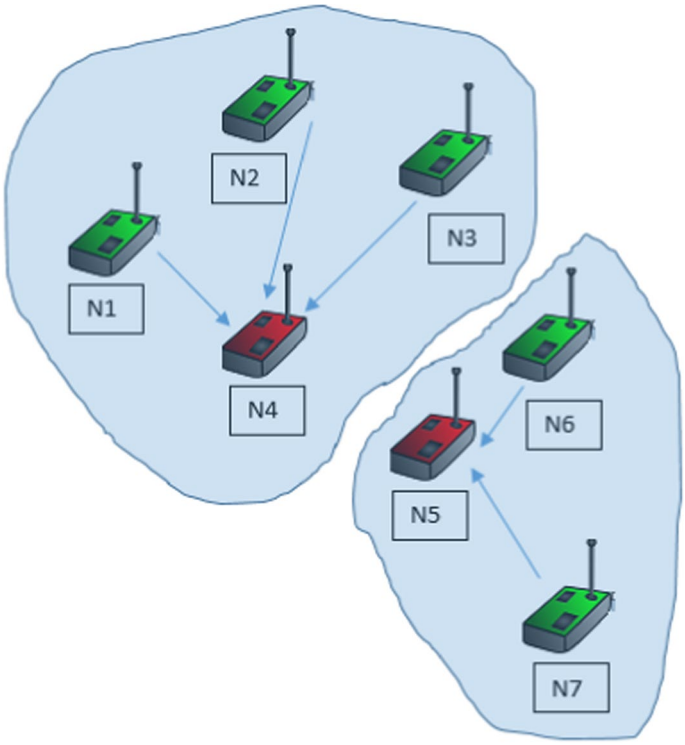


Fig. 3 Cluster head selection in case of overlapping sensing range

The first experiment is conducted to explore how the stability of clusters varies with network size. From Fig. 4 it is observed that the clusters become more stable when more nodes are deployed. Here, by number of rounds we mean the time after deployment as data are being transmitted after a fixed interval. The proposed algorithm is invoked whenever the residual energy of the cluster head is below a threshold energy, and eventually, a new cluster head is selected. When the number of deployed nodes is more, since there is more option of re-selection of cluster head, the clusters are more stable than when the number of nodes deployed is less. A network of 600 nodes and more indicates more or less stable clustering performance.

The next experiment is conducted to explore the effect of node densities and communication ranges on cluster formation as shown in Fig. 5. The figure reveals that the number of clusters formed is dependent on the communication range of the deployed nodes. Communication range of the nodes and the number of cluster formed are inversely proportional to each other. As according to the proposed algorithm, only one hop neighbours can remain in the same cluster; therefore, more the communication range, less would be the number of clusters, as one cluster can now cover a wider geographic area.

The next experiment is focused to explore the relation between the number of isolated nodes and number of rounds. From Fig. 6, it is observed that, initially, there are no isolated nodes. However, after running for longer time, there would

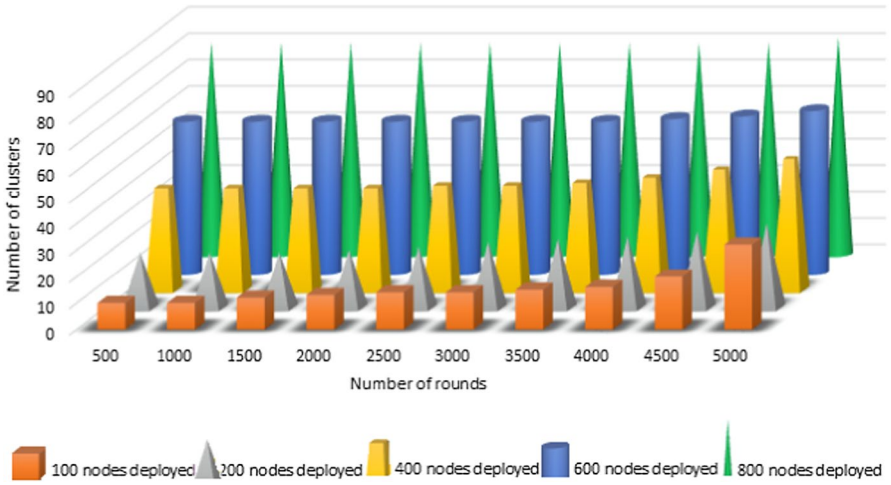


Fig. 4 Changes in number of clusters with respect to network size and simulation time

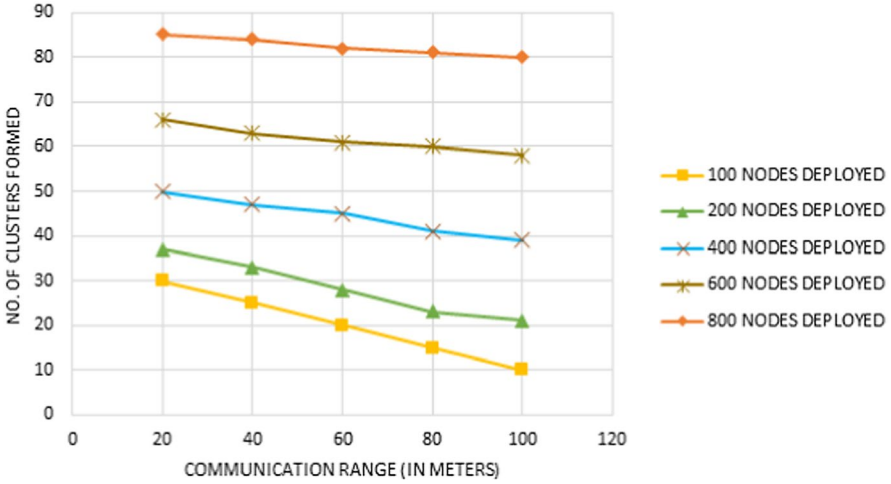


Fig. 5 Effect of sensors densities and communication ranges on number of clusters

be some isolated nodes as many neighbour nodes would be dead. It also depicts that percentage of isolated nodes are inversely proportional to the number of nodes deployed. A node is isolated only when all its one hop neighbours are dead. For a network of 800 nodes, even after 4500 rounds, that is, $4500/4 = 1125$ s after initiation (sending 4500 messages), there are hardly any isolated nodes. Even a sparse network of 100 nodes is able to operate successfully without any isolated nodes for 3000 rounds following the proposed algorithm.

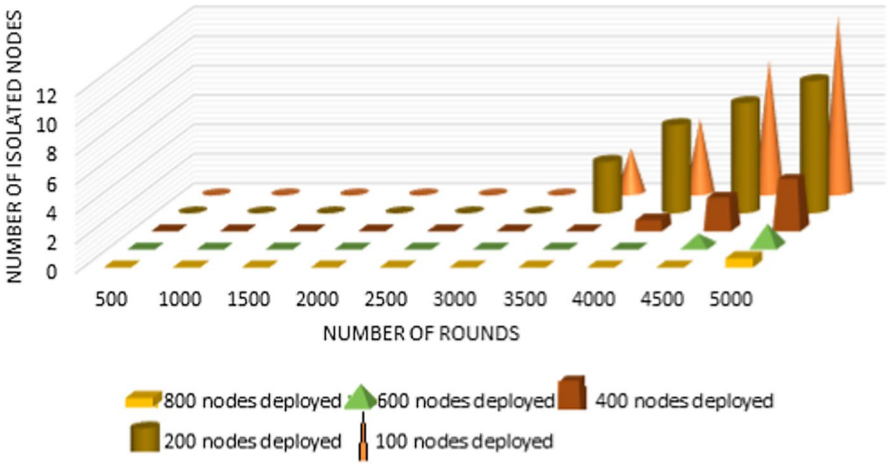


Fig. 6 Number of isolated nodes with respect to number of rounds

Consequently, the next experiment is conducted to explore the relation between the percentage of dead nodes and number of rounds. It is observed from Fig. 7 that the rate of increase in number of dead nodes is very low. Since the algorithm is using only one hop transmission, and re-selection of cluster heads ensures proper load balancing, hence, the life span of individual nodes gets improved.

In Figs. 8, 9, and 10, the percentage of isolated nodes, number of clusters formed, and percentage of dead nodes are tested with varying constants w, x, y, and z, respectively, where the weight coefficient of each evaluation factor satisfies

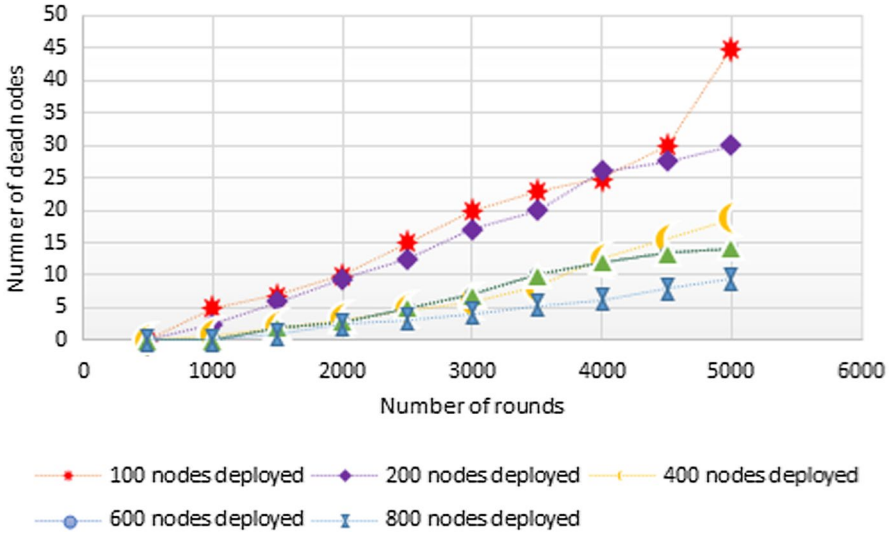


Fig. 7 Effect of percentage of dead nodes per round on number of nodes deployed

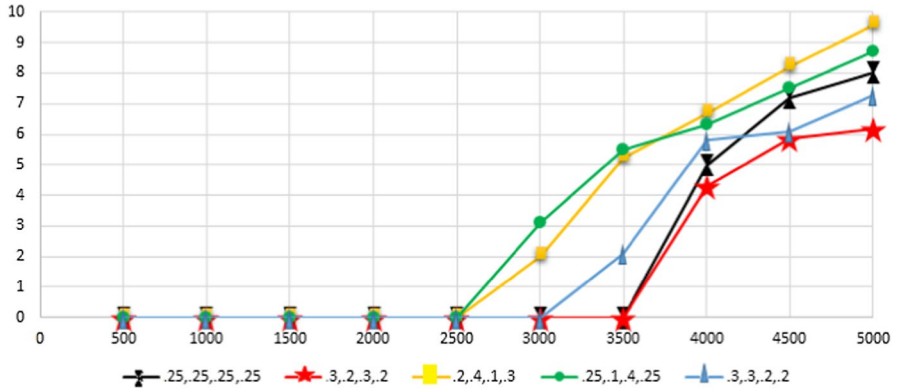


Fig. 8 Percentage of isolated nodes with respect to varying constants (w, x, y, z)

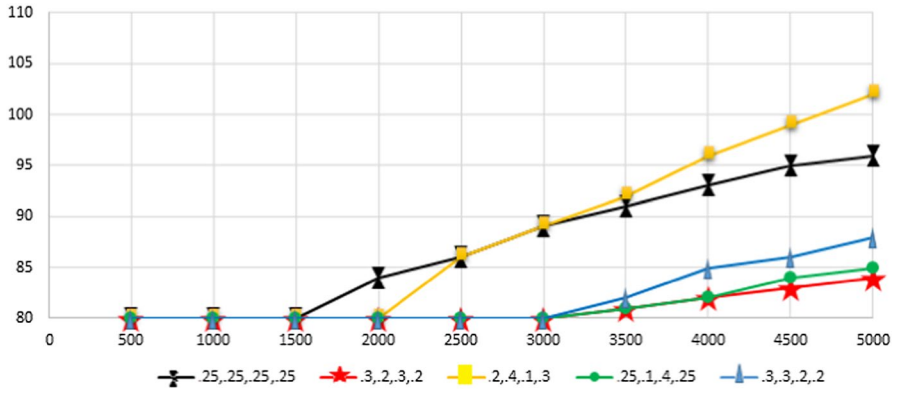


Fig. 9 Number of clusters formed with respect to varying constants (w, x, y, z), respectively

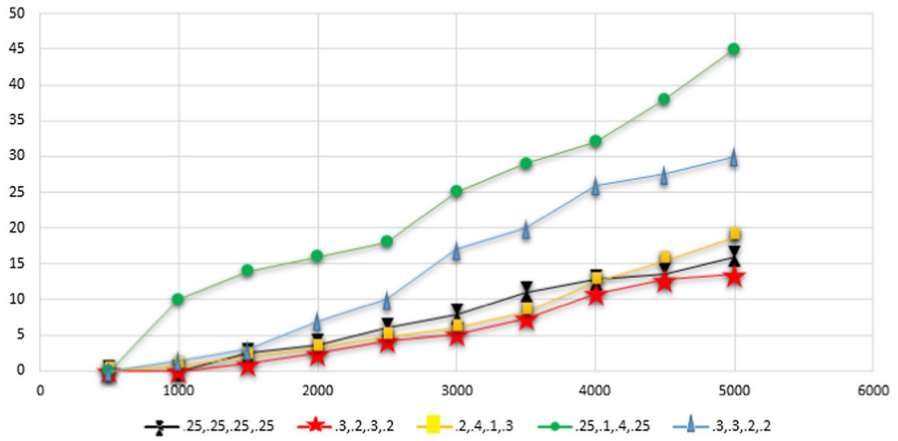


Fig. 10 Percentage of dead nodes with respect to varying constants (w, x, y, z), respectively

$w + x + y + z = 1$. Different representative value combinations are explored in order to cover the entire horizon. These constants are used in equation 3. As can be observed from the figures, the value combination of $w = 0.3, x = 0.2, y = 0.3, z = 0.2$ gives the best results as it results in more stable clusters (Fig. 9), resulting in lesser percentage of dead nodes (Fig. 10).

To compare the effectiveness of our protocol, our work is compared with the EPMS protocol [19] and UCRA-GSO protocol [20]. In EPMS, the authors have used particle swarm optimization and in UCRA-GSO authors have used GSO for clustering. The comparison is made on the basis of the number of nodes alive per round (NoS = 100). The results are depicted in Fig. 11. From the figure it is clear that, though in initial rounds the EPMS and UCRA-GSO are found to give comparable results, but, after around 10–20% nodes are dead, the proposed mechanism works better as it is changing the cluster heads depending on the residual energy. As time progresses, the improvement is even more apt.

It has been observed that after around 3000 rounds, the algorithm is re-invoked. The energy consumed by the nodes to rerun the algorithm is calculated, and residual energy of the nodes is updated accordingly by the simulation environment. Though it is obvious that additional energy is consumed while re-selecting the cluster heads, it is done only when the cluster head does not have the sufficient residual energy to perform its functions. In this scenario, the re-selection increases the lifetime of the network by balancing the load among the nodes. Because of this, load balancing among the number of alive nodes is better in our protocol.

Thus, to summarize our findings, the number of clusters is found to become stable for a long runtime if more than 600 nodes are deployed. More communication range would obviously call for lesser no of clusters. Denser networks provide better

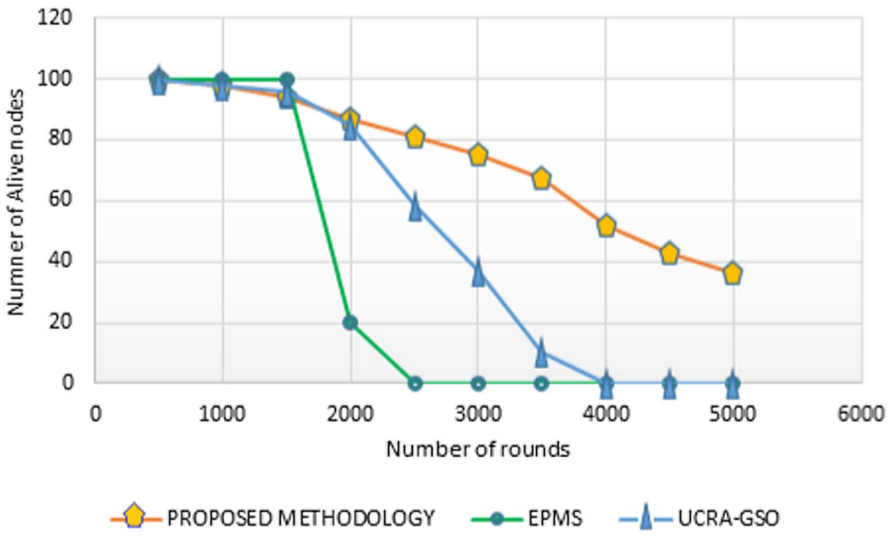


Fig. 11 Comparison of the proposed methodology with EPMS [19] and UCRA-GSO [20] with respect to number of alive nodes per round

performance. With 800 nodes, the network performs better without any isolated nodes even after 4500 rounds.

5 Conclusion

In this paper we have introduced an adaptive cluster head selection algorithm based on glow-worm swarm optimization. The algorithm is able to divide the whole network into optimised number of clusters. In the developed technique the number of nodes in each clusters is not fixed, and it automatically changes depending on the number of alive nodes in the network, which reduces the number of isolated nodes and also increases the lifetime of the network. The algorithm ensures minimum overlapping of the clusters using voting mechanism which minimizes the communication overhead. The algorithm is automatically initiated by the sink node when any cluster head is dead. Repeated selection of cluster heads and reformation of the clusters ensure proper load balancing in the network and simultaneously increase the lifetime of the network. The designed algorithm is executed in OMNET++ and has shown the various outcomes in different conditions such as after different number of rounds, changing the sensor densities, communication ranges and varying the constants. The results show how the clusters are self-adaptable after different number of rounds depending on number of alive nodes. Moreover, it has also been demonstrated how the developed technique works better than a state-of-the-art technology in terms number of alive nodes after a certain number of rounds.

We would like to discuss certain limitations of our proposed method. First, the proposed algorithm can tolerate node movement as long as the resultant topology remains unchanged. In future, we plan to explore such scenario where movement of some nodes in the clusters does not hamper the performance of the IoT devices. Second, heterogeneity of the nodes in terms of battery backup would not hamper the working of the algorithm. However, the performance could be improved if strategic placement of such nodes is considered. Thus, in future, placement of heterogeneous IoT nodes and their effect on clustering could be investigated. We plan to explore the strategic placement of edge servers for this purpose for better bandwidth and scalability.

References

1. Verdone R, Dardari D, Mazzini G, Conti A (2010) Wireless sensor and actuator networks: technologies, analysis and design. Academic Press, Cambridge
2. Xu L, O'Hare G, Collier R (2017) A smart and balanced energy-efficient multihop clustering algorithm (smart-beem) for mimo IoT systems in future networks. Sensors 17(7):1574
3. Sun X, Ansari N (2018) Dynamic resource caching in the IoT application layer for smart cities. IEEE Internet Things J 5(2):606–613
4. Subramaniyaswamy V, Manogaran G, Logesh R, Vijayakumar V, Chilamkurti N, Malathi D, Senthilselvan N (2019) An ontology-driven personalized food recommendation in IoT-based healthcare system. J Supercomput 75(6):3184–3216

5. Cheraghlou MN, Khadem-Zadeh A, Haghparast M (2019) EFT: novel fault tolerant management framework for wireless sensor networks. Wirel Person Commun 109:981–999. https://doi.org/10.1007/s11277-019-06600-x
6. Sun X, Ansari N (2017) Traffic load balancing among brokers at the IoT application layer. IEEE Trans Netw Service Manag 15(1):489–502
7. White G, Nallur V, Clarke S (2017) Quality of service approaches in IoT: a systematic mapping. J Syst Softw 132:186–203
8. Sathish Kumar J, Zaveri MA (2018) Clustering approaches for pragmatic two-layer IoT architecture. Wirel Commun Mob Comput 2018:16
9. Sung Y, Lee S, Lee M (2018) A multi-hop clustering mechanism for scalable IoT networks. Sensors 18(4):961
10. Jabeur N, Yasar AU-H, Shakshuki E, Haddad H (2017) Toward a bio-inspired adaptive spatial clustering approach for IoT applications. Future Gener Comput Syst 107:736–744. https://doi.org/10.1016/j.future.2017.05.013
11. Lee D, Lee HM (2018) Iot service classification and clustering for integration of IoT service platforms. J Supercomput 74(12):6859–6875
12. Geetha VA, Kallapur PV, Tellajeera S (2012) Clustering in wireless sensor networks: performance comparison of leach & leach-c protocols using ns2. Procedia Technol 4:163–170
13. Ding P, Holliday J, Celik A (2005) Distributed energy-efficient hierarchical clustering for wireless sensor networks. In: International Conference on Distributed Computing in Sensor Systems. Springer, pp 322–339
14. Liu X (2012) A survey on clustering routing protocols in wireless sensor networks. Sensors 12(8):11113–11153
15. Sari IRF (2017) Bioinspired algorithms for internet of things network. In: 2017 4th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE). IEEE, pp 1–1
16. Cui Z, Cao Y, Cai X, Cai J, Chen J (2019) Optimal leach protocol with modified bat algorithm for big data sensing systems in internet of things. J Parallel Distrib Comput 132:217–229
17. Sehgal A, Perelman V, Kuryla S, Schonwalder J (2012) Management of resource constrained devices in the internet of things. IEEE Commun Mag 50(12):144–149
18. Perera C, Jayaraman PP, Zaslavsky A, Christen P, Georgakopoulos D (2014) Mosden: an internet of things middleware for resource constrained mobile devices. In: 2014 47th Hawaii International Conference on System Sciences. IEEE, pp 1053–1062
19. Wang J, Cao Y, Li B, Kim H, Lee S (2017) Particle swarm optimization based clustering algorithm with mobile sink for WSNs. Future Gener Comput Syst 76:452–457
20. Yong L, Mufang H, Ke Z, Renrong X, Xiuwu Y, Qin L (2019) Uneven clustering routing algorithm based on glowworm swarm optimization. Ad Hoc Netw 93:101923
21. Puschmann D, Barnaghi P, Tafazolli R (2016) Adaptive clustering for dynamic IoT data streams. IEEE Internet Things J 4(1):64–74
22. Lyu L, Jin J, Rajasegarar S, He X, Palaniswami M (2017) Fog-empowered anomaly detection in IoT using hyperellipsoidal clustering. IEEE Internet Things J 4(5):1174–1184
23. Fredj SB, Boussard M, Kofman D, Noirie L (2013) A scalable IoT service search based on clustering and aggregation. In: 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing. IEEE, pp 403–410
24. Fan Q, Ansari N (2018) Application aware workload allocation for edge computing-based IoT. IEEE Internet Things J 5(3):2146–2153
25. Salman O, Elhajj I, Chehab A, Kayssi A (2018) IoT survey: an SDN and fog computing perspective. Comput Netw 143:221–246
26. Javaid N, Cheema S, Akbar M, Alrajeh N, Alabed MS, Guizani N (2017) Balanced energy consumption based adaptive routing for IoT enabling underwater wsns. IEEE Access 5:10040–10051
27. Yan S, Peng M, Cao X (2018) A game theory approach for joint access selection and resource allocation in UAV assisted IoT communication networks. IEEE Internet Things J 6(2):1663–1674
28. Krishnanand KN, Ghose D (2009) Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. Swarm Intell 3(2):87–124
29. Varga A (2019) A practical introduction to the omnet++ simulation framework. In: Recent Advances in Network Simulation. Springer, pp 3–51

30. Udugama A, Förster A, Dede J, Kuppusamy V (2019) Simulating opportunistic networks with OMNET++. In: Recent Advances in Network Simulation. Springer, pp 425–449
31. Olaleye OG, Ali A, Perkins D, Bayoumi M (2018) Modeling and performance simulation of PULSE and MCMAC protocols in RFID-based IoT network using OMNeT++. In: 2018 IEEE International Conference on RFID (RFID). IEEE, pp 1–5
32. Ariza A, Inzillo V (2019) Inetmanet framework. In: Recent Advances in Network Simulation. Springer, pp 107–138

The Journal of Supercomputing (2023) 79:18227–18255
https://doi.org/10.1007/s11227-023-05358-1

Cuckoo search optimization-based energy efficient job scheduling approach for IoT-edge environment

Mohana Bakshi[1,2] · Chandreyee Chowdhury[1] · Ujjwal Maulik[1]

Accepted: 27 April 2023 / Published online: 13 May 2023

Abstract

Recently developed IoT devices are capable of gathering, storing, and processing more data than ever before. This calls for the need for scalability. Through the use of edge computing, more processing functions can be relocated closer to where the data is gathered through the IoT devices. Here, processing tasks may be placed in the edge computing units (ECUs). Each ECU may host a cloudlet consisting of a number of virtual machines, where tasks could be executed. Due to the need to ensure near real-time response of the jobs, efficient job scheduling is required. Few recent works addressed this issue of job scheduling at the edges. However, many important constraints such as job dependency, job conflict along with heterogeneous edge infrastructure are not found to be considered. Accordingly, in this paper, we have proposed an optimal job scheduling approach based on the cuckoo search algorithm to handle these challenges subject to energy efficiency and resource utilization. The proposed algorithm is simulated and found to work notably well as compared to state-of-the-art edge-computing-based job scheduling techniques, especially when the jobs have conflict or dependency among them. The work is reported to achieve above 85% resource utilization even in the presence of job conflicts and dependencies.

✉ Chandreyee Chowdhury
chandreyee.chowdhury@gmail.com

Mohana Bakshi
mohanaprofile@gmail.com

Ujjwal Maulik
ujjwal.maulik@jadavpuruniversity.in

[1] Jadavpur University, Kolkata, India

[2] Kendriya Vidalaya Sangathan, Kendriya Vidyalaya Burdwan, Burdwan, India

Keywords IoT · Cuckoo search optimization · Job scheduling · Energy efficiency · Resource utilization · Job conflict · Job dependency

1 Introduction

In this digital era, Internet of Things (IoT) [1, 2] is an emerging field that connects various smart devices and sensing appliances to the Internet such that a plethora of societal applications can be designed for smart agriculture, smart grid, smart home, smart healthcare, and so on. This leads to the wide availability of networked computing environments for private, public, and business spaces. Most of the IoT applications are data intensive requiring huge processing and storage capability. Thus, cloud computing came up as the default solution for large-scale and moderate-scale IoT-based applications [3][4]. IoT with cloud computing infrastructure is particularly utilized when the applications require high availability and processing, and need ample storage space [5, 6].

The extended capability of IoT enables data collection, data storage, and data analytics closer to the end-user so that it can be made suitable for time stringent real-time applications [7]. However, the decentralized nature of IoT does not scale well with the rather centralized structure of the cloud. The requirement of huge network bandwidth can be a hindrance to real-time IoT cloud-based applications. Maintaining a steady quality of service along with the quality of experience for heterogeneous IoT users is challenging [5, 8–10]. In this scenario, the edge computing paradigm boosts IoT applications by eliminating offloading of information to or from the cloud over the Internet. Edge computing is used to process time-sensitive data, while cloud computing is used to process data that is not time-driven. Therefore, the timeliness of task execution is more crucial in edge computing than in cloud computing. Edge Computing is regarded as ideal for operations with extreme latency concerns. The edge devices are generally placed at the gateway in close proximity to the end devices. Thus, real-time services can be designed efficiently while taking advantage of location awareness as edge devices are nearer to the source. Due to the wide availability of different types of portable computing devices and servers, edge computing-based solutions are highly investigated recently [11–13].

Placing the IoT services in the Edge Computing Units (ECUs) definitely mitigates the bandwidth burden in IoT and meanwhile improves the quality of service (QoS) [14–17]. However, as the service requirements are originated from a variety of IoT devices, characterized by various data standards, the IoT service placement in edge computing is somewhat critical [18–20].

Considering the geographically scattered IoT services in the whole deployment area, the ECUs are subject to underload or overload resource usage, which affects the service performance in edge computing [21, 22]. Furthermore, due to the wide deployment of ECUs, the energy cost needs to be optimized for the sustainable development of the IoT industry [23, 24].

On the other hand, some IoT services may have conflicts, and hence, they may not be processed in the same ECU due to security issues. Apart from conflicts,

dependencies can also exist among the IoT services. Thus, the order of execution must be maintained [25] at the edge so that the output of a task can be fed as an input to its dependent task. Hence, for providing proper QoS of IoT services across ECUs, it is of utmost significance to avoid privacy conflicts and maintain the job dependencies among the IoT services during IoT service placement [26–28]. Priority is another aspect that should be looked into in order to address the different QoS requirements of the various IoT services. Particularly, the real-time services should be assigned high priority in order to retain the QoS guarantee [29–31].

With these observations, it is still challenging to optimize the overall ECU execution performance w.r.t the resource usage and the power consumption of ECUs for IoT service placement problems. There are very few works on job scheduling in IoT-based edge computing environments. Most of the works are found to consider one aspect or the other. Some works are based on job priority as in [9] while the work reported in [23] considers resource utilization. However, works could not be found that consider the following important aspects.

- Job priority,
- Inter Job dependency,
- Job conflicts, and
- Heterogeneous ECU capacities.

Accordingly, the main contribution of this work is that a job scheduling algorithm is proposed considering all the above conditions that produce a job schedule in a way to optimize the overall ECU performance in terms of both the resource utilization and energy efficiency. In view of these challenges, the candidate solution space becomes exponentially large. This motivates the usage of meta-heuristic optimization algorithms as they enable exploration and exploitation across the search space. Cuckoo search-based optimization is found to be efficient in reaching convergence, while it also has a powerful exploration mechanism through levy flights. Hence, we have proposed a cuckoo search-based Job Scheduling (CSJS) approach in this paper that takes care of the four above-mentioned conditions while producing an optimized job schedule at the edge devices.

The rest of the paper is organized as follows: Sect. 2 presents the literature survey followed by a brief description of the proposed methodology in Sect. 3. Section 4 summarizes the experimental setup and analysis of the experimental results while Sect. 5 concludes.

2 Related work

In this section, the existing job scheduling protocols applicable in the field of IoT are discussed briefly. The works also highlight the need for edge computing research w.r.t IoT job scheduling as its characteristics differ from that of cloud.

In [23], authors utilized the edge computing as a novel paradigm to process the IoT service placement in smart cities. They considered conditions such as, load

balance, energy consumption and resource utilization to optimize the quality of IoT services in smart city. They have adopted the strength Pareto evolutionary algorithm (SPEA2) technique to obtain the balanced service placement strategies for optimizing multiple performance metrics while avoiding the privacy leakage and satisfying the time constraints. The authors did not consider the dependency and priority constraint among jobs. In the study the job placement is given, job scheduling is not considered.

In a simulation study, the authors of [1] suggested a conceptual framework for fog resource provisioning. They introduced the concept of "fog cell", which is a software component running on fog devices that controls and monitors a particular group of IoT devices. Using this and other related concepts, they model orchestration of IoT devices using a hierarchical cloud/fog resource control and provide a suitable resource provisioning solution for distributing tasks among them. The study in [24] has proposed a messaging method with low overhead that notifies fog nodes about nearby replica nodes so that the replica nodes can be used for handling requests instead of depending on cloud storage. However, energy consumption was not considered in both the study.

In [26], a Simulated Annealing Algorithm (SAA) is used to compare the performance of the GA in its centralized and distributed forms. SAA is another AI-based algorithm which can be used to solve NP complete optimization problems in a heuristic approach. They haven't considered any dependency and conflict constraint among the jobs.

In [27], authors first introduced FOGPLAN, a framework for QoS-aware Dynamic Fog Service Provisioning (QDFSP). QDFSP concerns the dynamic deployment of application services on fog nodes, or the release of application services that have previously been deployed on fog nodes, in order to meet the low latency and QoS requirements of applications while minimizing the cost. FOGPLAN framework is practical and operates with no assumptions and minimal information about the IoT nodes. Next, they have presented a possible formulation (as an optimization problem) and two efficient greedy algorithms for addressing the QDFSP at one instance of time. Finally, the FOGPLAN framework is evaluated using a simulation based on real-world traffic traces. Simulation results showed that delay and overall cost is reduced but at the cost of slower runtimes.

In [29], authors designed a novel mobility-aware online service placement framework to achieve a desirable balance between time-averaged user-perceived latency and migration cost. To tackle the unavailable future system information, which involves mobility pattern and request arrival processes, authors utilized Lyapunov optimization technique to incorporate the long-term budget into a series of real-time optimization problems. They have developed two efficient heuristic schemes based on the Markov approximation and best response update techniques to approach a near-optimal solution. Through extensive simulation, authors demonstrated the effectiveness of their online algorithm while maintaining the long-term migration cost constraint. However, in this work authors have considered only mono-objective optimization in terms of latency. Energy consumption or resource utilization was not considered in this study.

In [30], authors presented two alternatives for the service orchestration in such distributed edge systems. They proposed strategies either to orchestrate the system in a completely flat architecture, or in a hierarchical recursive manner. The first option assumes that all the infrastructure islands are controlled by a single manager, i.e., OpenStack, so an extension is proposed to make it suitable for the distributed edge topology. The second option places an extra orchestration component next to each infrastructure manager, e.g., next to an edge node's Docker engine, and organizes them in a multi-layer topology. With both solutions the aim is to quickly and efficiently map incoming service deployment requests to physical resources. However, in this study, authors have only considered service placement and disregarded server activation and deactivation.

The authors in [32] proposed a PSO-based heuristic strategy to solve the joint problem of service placement and task provisioning. This algorithm solved the resource scheduling problem by greedy-based and genetic-based algorithms. However, scarcity of resources in fog nodes, due to the main functions load, significantly degrades the platform's performances.

In [33], authors have mathematically formulated the task scheduling problem to minimize the total energy consumption of fog nodes (FNs) while meeting the quality of service (QoS) requirements of IoT tasks. They have also considered the minimization of the deadline violation time in their model. Next, authors have proposed two semi-greedy algorithms to efficiently map IoT tasks to the FNs. It is interesting that they have not only considered the deadline requirement but also evaluated the performance based on the total deadline violation time.

In [34], the task scheduling problem in fog-based IoT applications is investigated with the aim of achieving an efficient policy in terms of time and energy saving under resource and deadline constraints. In order to ensure efficient scheduling, authors have proposed and utilized CDDQL scheduling algorithm. To deal with long-time waiting tasks in the VM queue, they have proposed an approach to reschedule them. Having more than one scheduler helped in load balancing and avoided the Single Point of Failure (SPoF) issue in the network. According to the results, the algorithm performed better than other algorithms including first come first serve strategy, random scheduling strategy and a learning-based Q-learning scheduling. Having more than one scheduler helped in load balancing and avoided the SPoF issue in the network.

A nature-inspired multi-objective task scheduling algorithm called the electric earthworm optimization algorithm (EEOA) is proposed in [35] for IoT requests in a cloud-fog framework. Based on execution time, cost, makespan, and energy consumption, the suggested scheduling technique's performance was assessed on a simulator using significant instances of real-world workloads.

In [36], authors have proposed a dual-phase metaheuristic algorithm called CSSA-DE to efficiently assign an IoT-task at a VM in the cloud. First, they have conducted a clustering approach to group computing nodes into effective clusters. Then, authors have integrated the sparrow search algorithm (SSA) with the differential evolution (DE) algorithm to expand the high search efficiency of finding an appropriate pair task-VM combination. The work takes care of resource fragmentation and

Table 1 Comparison with state of the art technologies

Work and year of publication	Technique used	Resource allocation	Energy conservation	Job dependency	Job conflict	Priority
[1] in 2017	GA	✓	✗	✗	✗	✗
[24] in 2018	RPA	✓	✗	✗	✗	✗
[27] in 2019	Greedy Algorithm-based FOGPLAN	✗	✓	✗	✗	✗
[32] in 2019	PSO-based heuristic strategy	✓	✗	✗	✗	✗
[23] in 2020	SPEA2	✓	✓	✗	✓	✗
[34] in 2021	CDDQL scheduling algorithm	✓	✓	✗	✗	✗
[33] in 2022	Priority-aware Semi-Greedy algorithm (PSG)	✗	✓	✗	✗	✓
[36] in 2023	Dual-phase metaheuristic algorithm	✓	✓	✗	✗	✗
[35] in 2023	Electric earthworm optimization algorithm (EEOA)	✗	✓	✗	✗	✗
Proposed work (CSJS)	Cuckoo Search	✓	✓	✓	✓	✓

hence utilization. However, job dependency and job conflict are not considered by any of the above mentioned ([33–36]) works.

In Table 1, the comparison with the proposed algorithm along with the state of the art technologies have been shown. As can be observed, the existing works indicate usage of meta-heuristics techniques to optimize the edge performance. Conditions such as, QoS, migration cost, and resource utilization are given importance. Most of the works mainly considered the server side issues while framing the optimization approach. The deadline requirements of the tasks, heterogeneous ECU environments, and job priorities are considered by a few recent works. However, most of the literature often ignored any relationship such as, dependency or conflict in the jobs to be scheduled. However, these are important criteria for scheduling jobs at the edge nodes. Thus, a meta-heuristics-based approach is designed in this work that is based on cuckoo search optimization technique. An overview of this technique is detailed in the following section.

3 Overview of cuckoo search algorithm (CSA)

CSA is one of the latest nature-inspired metaheuristic algorithms, developed in 2009 by Xin-she Yang and Suash Deb [37]. CS is based on the obligate brood parasitic behavior of some cuckoo species in combination with the Levy flight behaviour of cuckoo bird. The pseudocode for cuckoo search is described in Algorithm 1.

Algorithm 1: CSA

```
Objective function f(x);
Generate initial population of n host nest;
Evaluate fitness and rank eggs;
while t > MaxGenerationorStopcriterion do
    t=t+1;
    Get a cuckoo randomly/ generate new solution by Levy flights;
    Evaluate quality/fitness F_i;
    Choose a random nest j;
    if (F_i > F_j) then
        Replace j by the new solution;
    end
    Abandon a fraction (pa) of worse nests;
    build new ones at new locations via Levy flights;
    Keep the best solutions (or nests with quality solutions);
    Rank the solutions and find the current best;
end
Post process results and visualization
```

When generating new solutions $x_{\alpha}^{(t+1)}$ for αth cuckoo, a Levy flight is performed using the following equation

$$x_{\alpha}^{(t+1)} = x_{\alpha}^{t} + \alpha s \bigotimes H(p_a - \varepsilon) \bigotimes (x_j^t - x_k^t) \tag{1}$$

where x_j^t and x_k^t are two different solutions selected randomly by random permutation, H(u) is a Heaviside function, ε is a random number, and s is the step size.

On the other hand, the global random walk is carried out by using Levy flights

$$x_{t+1}^{i} = x_t^i + \alpha L(s, \lambda) \tag{2}$$

$$L(s, \lambda) = \frac{\lambda \Gamma(\lambda) sin(\pi \lambda / 2)}{\pi} \left(\frac{1}{s^{1+\lambda}} \right), (s \geqslant s_0 > 0) \tag{3}$$

where $\Gamma(\lambda)$ is the Γ function and λ the random step length. Levy flights essentially provide a random walk while the random steps are drawn from a Levy distribution for large steps ($1 < \lambda \leq 3$).

The three idealized rules of cuckoo search and how these rules help in solving the optimization problem are listed in Table 2.

CSA has mainly two advantages

a. It satisfies the global convergence requirements.
b. It supports local and global search capabilities.

Recent studies show that in comparison with Genetic Algorithm, cuckoo search has a high exploration ability due to its mutation related Levy flights, yet can converge quickly for job scheduling at grids as in [38].

CSA has two distinct advantages, such as, efficient random walks and balanced mixing. Levy flights are reported to be applied in optimization to improve the efficiency of the search process for nature inspired algorithms as in [39]. In [40], authors have proved that CSA can degenerate into a variant of Differential Evolution (DE), as well as an Accelerate Particle Swarm Optimization (APSO) algorithm. This means that SA, DE and APSO are special cases of CSA and that is one of the reasons that CSA is so efficient.

In this work, CSA has been applied for scheduling the jobs at the edges.

We hypothesize that consideration of QoS-based constraints (such as, priority, job conflict, and job dependency) in the fitness function of a metaheuristic algorithm like cuckoo search which satisfies global convergence requirements as well as local and global search capabilities will converge to come up with an optimized schedule for the IoT jobs.

This hypothesis has been validated through algorithm formulation and experimentation as detailed in the subsequent sections.

Table 2 Rules of cuckoo search

Rule no.	Rule	How does it help in solving the optimization problem
i	Each cuckoo lays one egg at a time, and dumps its egg in a randomly chosen nest	The new cuckoo egg intends to replace a previous egg by improving the quality of solutions
ii	The best nests with high quality of eggs will carry over the next generation	The "selection of best" finds the best local optimum among the cuckoo population at each generation, progressing the convergence of the cuckoo to a feasible global optimum
iii	Each cuckoo egg laid in a host nest has an assigned discovery rate value, p_a, and when a predefined discovery condition is met, the host bird can either build a new nest or just get rid of the cuckoo egg	A completely new solution far from current population's solution is generated enabling the global scale exploration

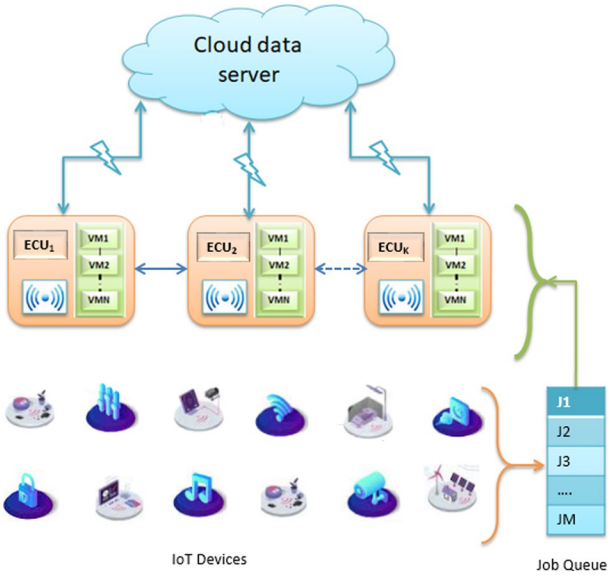


Fig. 1 System model

4 Proposed methodology

The architecture and system model is presented first in order to formally define the problem. Then, the proposed job scheduling approach is detailed in this section.

4.1 System model

The IoT service placement architecture utilized in this work is presented in Fig. 1. The architecture consists of three layers.

1. IoT device layer
2. ECU layer
3. Cloud Data Server layer

In IoT device layer, numerous IoT devices are placed. Here, M number of IoT devices are considered to be deployed. Let each IoT device deploys one service that needs to be processed remotely. That means M number of jobs are to be processed, defined as $J = \{j_1, j_2, \ldots, j_M\}$. Initially, jobs are placed in a waiting queue. Proposed scheduling algorithm would determine which jobs should be placed in which ECU depending upon various criteria so that job execution cost is minimized whereas the ECU utilization would be maximized.

Table 3 Used parameter details

Parameters for defining ECU	
VM_i	Number of virtual Machines in ith ECU
$jVM_{i,r}$	List of jobs placed in ith ECU w.r.t round r
L	Number of ECU
Parameters for defining Jobs	
M	Number of jobs
DUR_j	Execution time of jth job
RVM_j	Required number of VM for execution of the jth job
P_j	Priority sequence number of the jth job
S_j	State of the jth job. Defined in Eq. 4
D_j	Dependency list of the jth job, i.e., list of all jobs j_k such that $D_j^k = 1$
	Defined in Eq. 5
C_j	Conflict list of the jth job, i.e., list of all jobs j_k such that $C_j^k = 1$
	Defined in Eq. 7
RD_j	Execution ready state of the job. Defined in Eq. 6
Parameters for energy cost calculation	
α	Power required for active ECU
β	Power required for active VMs
γ	Power required for inactive VMs in active ECUs
$AE_{i,r}$	Status of ith ECU w.r.t round r. Defined in Eq. 8
$AVM_{i,j,r}$	Status of jth VM of ith ECU w.r.t round r. Defined in Eq. 9
TE_r	Total energy consumed in round r. Defined in Eq. 10
NE_r	Normalized energy consumed in round r. Defined in Eq. 11
T	Total time for execution of all jobs
EC	Total energy consumed in round r. Defined in Eq. 12
Parameters for resource utilization calculation	
NRU_r	Normalized resource utilization in round r. Defined in Eq. 13
Parameters to calculate fitness value	
ζ	Constant multiplied with energy parameter
η	Constant multiplied with resource utilization parameter
$CJ_{i,j,r}$	List of compatible jobs of job j in ith ECU w.r.t round r

In the ECU layer, there are L number of ECUs deployed where $L << M$, defined as $E = \{E_1, E_2, \ldots, E_L\}$. Each ECU consists of an access point and a variable number of VMs. Jobs will be placed in the VMs of the respective ECUs according to

the scheduling algorithm, such that the energy consumption cost is minimized and simultaneously, the resource utilization is maximized. It is the responsibility of the ECU to execute the jobs or to send them to cloud data server as per the need.

All the parameters related to job definition, ECU definition, energy cost and resource utilization have been described in Table 3.

4.2 Problem definition

The problem can be defined as follows.

Let there be M number of jobs and L ECUs. Each ECU has a variable number of VMs. The objective of the algorithm is to produce a job schedule such that

a. Energy consumption is minimized.
b. Resource utilization is maximized.
c. In the schedule, properties like job conflict and dependency are maintained.
d. Prioritizing the execution of higher priority, ready for execution jobs.

Some of the constraints are

1. All jobs are non-pre-emptive.
2. Any single job will be allocated to a single ECU, though it may require more than one VMs.
3. Any job can start its execution only after all the jobs, on which it is dependent have already completed their execution.
4. No two jobs can be placed in the same ECU at the same time which have conflicts among themselves.

To solve the above job scheduling optimization problem, a cuckoo search-based job scheduling algorithm has been proposed.

4.3 Method overview

All the jobs $\{j_1, j_2, \ldots, j_M\}$ are placed in the priority queue, with status 0 as defined in Eq. 4, sorted according to their respective priority index $\{p_1, p_2, \ldots, p_M\}$.

$$s_i = \begin{cases} 0 & \text{if } j_i \text{ is in the waiting queue} \\ 1 & \text{if } j_i \text{ is placed in the ECU} \\ 2 & \text{if } j_i \text{ has been completed and removed from the ECU} \end{cases} \tag{4}$$

Initially, all the VMs of all the ECUs $\{E_1, E_2, \ldots, E_M\}$ are empty. A job j_i is ready to be executed if all the jobs j_k on which it is dependent, as defined in Eq. 5, have been executed.

$$D_i^k = \begin{cases} 1 & \text{if } j_i \text{ is dependent on } j_k \\ 0 & \text{if } j_i \text{ is not dependent on } j_k \end{cases} \tag{5}$$

The scheduling algorithm extracts the list of highest priority jobs which are ready to be executed, i.e $RD_j = 1$ as per equation 6.

$$RD_j = \begin{cases} 1 & \forall \text{jobs } j_p \text{ if } D_j^p = 1 \text{ and } s_p = 2. \\ 0 & \text{Otherwise} \end{cases} \tag{6}$$

Then, a set of initial solutions has been generated with all combinations of ready to be executed, highest priority jobs and the ECUs. Now using these initial solutions, a cuckoo search-based algorithm is executed to select the set of solutions with the highest fitness value. To calculate the highest fitness value, at first, the compatibility of the ECUs with the jobs in the solution is checked. A job is compatible with an ECU if the ECU has sufficient number of available VMs required for the execution of the job and no already assigned jobs have conflict with the job under consideration, as defined in the Eq. 7.

$$C_i^k = \begin{cases} 1 & \text{if } j_i \text{ has conflict with } j_k \\ 0 & \text{if } j_i \text{ does not have conflict with } j_k \end{cases} \tag{7}$$

In the proposed work, in every round, the energy cost and resource utilization have been calculated that are fed in the fitness value calculation so that the optimized job sequence can be selected. The energy cost and the resource utilization calculations are shown in the following subsections.

4.3.1 Energy cost calculation

For a job that is compatible with an ECU in the solution set, the amount of energy to be consumed, if the job is placed in that ECU, is calculated. The number of active ECUs with active VMs are calculated as follows.

$$AE_{i,r} = \begin{cases} 1 & \text{if } len(JVM_{i,r}) > 0 \\ 0 & \text{Otherwise} \end{cases} \tag{8}$$

Equation 8 determines whether the ith ECU is active in round r. The ith ECU is considered to be active in round r if there is at least one VM in that ECU, in that particular round, which is executing a job. That means the length of $JVM_{i,r}$ must be greater than 0, where $JVM_{i,r}$ is the list of jobs placed in the ith ECU w.r.t round r.

Whether the jth VM of ECU_i is active in round r is determined as follows.

$$AVM_{i,j,r} = \begin{cases} 1 & \text{if } len(JVM_{i,r}[j]) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

$JVM_{i,r}$ is the list of jobs of ith ECU in round r. For example, if the ECU_i has 3 VMs then $JVM_{i,r}$ may contain the list [[j1], [], [j2]]. That means in the round r, $VM_{i,1}$ is executing j_1, while $VM_{i,2}$ is not executing any job, and $VM_{i,3}$ is executing j_2. So, $JVM_{i,r}[1] > 0$, $JVM_{i,r}[2] = 0$, and $JVM_{i,r}[3] > 0$. Thus, by Eq. 9, $AVM_{i,1,r} = 1, AVM_{i,2,r} = 0$, $AVM_{i,3,r} = 1$. The total energy consumed in round r is calculated using Eq. 10. The energy is consumed in three ways, through active ECUs (α per unit), active VMs (β per unit), and inactive VMs of the active ECUs (γ per unit). Accordingly, the total energy requirement, TE_r is calculated as follows.

$$TE_r = \alpha \times \sum_{p=1}^{i} \left(AE_{p,r}\right) + \beta \times \sum_{i=1}^{L} \sum_{j=1}^{VM_i} \left(AVM_{i,j,r}\right) + \gamma \times \sum_{i=1}^{L} AE_i \times \sum_{j=1}^{VM_i} (1 - AVM_{i,j,r}) \tag{10}$$

The total energy cost, TE_r is normalized w.r.t the maximum energy consumption per round as follows.

$$NE_r = \frac{TE_r}{(L \times \alpha) + \beta \times \sum_{p=1}^{L} VM_p} \tag{11}$$

Accordingly, the total energy consumption for T rounds is computed as follows.

$$EC = \sum_{r=1}^{T} TE_r \tag{12}$$

4.3.2 Resource utilization

The normalized resource utilization is calculated, assuming the job to be placed in that ECU as follows.

$$NRU_r = \frac{\sum_{i=1}^{L} \sum_{j=1}^{VM_i} AVM_{i,j,r}}{\sum_{i=1}^{L} (AE_{i,r} \times VM_i)} \tag{13}$$

It represents ratio of the total number of active VMs with the total number of VMS in all active ECUs. Any active ECU consumes a lot of energy irrespective of how many jobs are placed there. So, the proposed algorithm tries to maximize the utilization of the VMs of the active ECUs before needing to activate an additional ECU. Since the inactive ECUs do not consume any energy therefore, the normalized resource utilization has been calculated taking only the number of active ECUs. The algorithm is detailed in the following section.

4.3.3 Algorithm overview

The proposed cuckoo search-based Job Scheduling (CSJS) algorithm takes the list of jobs to be executed and returns the optimized job sequence such that the consumption of energy cost is minimized and the resource utilization is maximized.

Since there are M jobs and L ECUs so there will be M^L possible ways of allocation of such M jobs in L ECUs. The jobs considered for execution must have the following properties.

a. The job has not started execution.
b. There exists at least one ECU such that it has sufficient number of empty VMs for execution of the job.
c. All the jobs on which the opted job is dependent have completed their execution.

The suitability of ECU for the execution of the job depends on mainly two factors.

a. There are enough VMs available in the ECU for the execution of the job.
b. No job which has a conflict with the selected job is presently being executed in the ECU.

The CheckCompatibility(job_j, ecu_i) procedure is invoked to check the compatibility between an ECU and a job. It returns 1 if job_j is compatible with ecu_i, otherwise, returns 0. This procedure is invoked by the fitness function F(job_j, ecu_i), while calculating the fitness value of a solution.

Depending on the fitness function, the proposed CSJS algorithm, summarized as Algorithm 2, selects the best suitable solution. For implementation of cuckoo search, on Steps 9 to 19 of Algorithm 2, the initial nests are considered to be randomly generated sequence of (job,ECU) set. While generating new solutions in

step 16, for say cuckoo *i*, a Levy flight is performed. Here, the consecutive jumps/steps of a cuckoo essentially form a random walk process which obeys a power-law step-length distribution with a heavy tail. Thus, the chances of stucking at a local optima are minimized. As *t* increases, steps started decreasing and intensification of solution space has been achieved. From here the algorithm would find out the best cuckoo, that is the optimized sequence of (job, ECU) set. The cuckoo Search is executed either up-to a given no of times or till three iterations that the algorithm chooses the best cuckoo, whichever is less. According to the result produced by the algorithm, the selected jobs are placed to the respective ECUs and the status of the jobs are changed to 2 following Eq. 4. The procedure is repeated till all the VMs are full or there is no ready to be executed job whose status = 0. The jobs start executing. When a job finishes its execution, it is removed from all the VMs which it is acquiring and its status becomes 2. Whenever a VM becomes empty, the procedure of job selection is again repeated, till all the jobs have status= 2. The fitness value is calculated by Algorithm 3 where the normalized resource consumption and resource utilization are taken into account while assigning the jobs to compatible ECUs. The weight for the two factors are determined empirically. It is to be noted that the energy consumption is a cost that should be minimized while resource utilization is the benefit that should be maximized.

Table 4 Experimental setup parameters

Key parameter	Value
M	250
L	10
VMi	[1, 7]
α	300 W
β	50 W
γ	30 W

Algorithm 2: CSJS

Input: Job Queue
Output: Optimized job sequence

Initially all jobs are placed in a waiting queue J;
J is sorted according to priority ;
r=1 ;
F_j=0 ;
while $\exists j_p \epsilon J : s_p! = 2$ **do**
  if $(\exists E_i \epsilon E : length(JVM_{i,r}) < VM_i) \& (\exists j_p : RD_p == 1 \& s_p == 0)$ **then**
    Pick the list of jobs j_H ready for execution;
    Generate initial soln (set of nests) with all combinations of jobs $\epsilon j_H \& E_i \epsilon ECU$;
    while *(t<MaxGeneration) or (stop criterion)* **do**
      Get a Cuckoo i randomly via Levy flights;
      Evaluate its quality/fitness F_i;
      if $(F_i > F_j)$ **then**
        Replace j by the new solution;
      end
      Abandon a fraction (pa) of worse nests;
      build new ones at new locations via Levy flights;
      Keep the best solutions (or nests with quality solutions);
      Rank the solutions and find the current best;
    end
    Insert the selected Job j_s to the selected EVM E_q;
    Put $s_s = s_s + 1$
  else
    Start executing the jobs s.t $s_p = 1$ i.e; already placed in VMs;
    r=r+1;
    if *job j_p is completed* **then**
      remove the job from all VMs which it is acquiring;
      Put $s_p = s_p + 1$;
      continue;
    end
  end
end

Table 5 Jobs description for performing a case study

id	Priority	RVM	Dependency	Conflict	Duration
0	6	1	[]	[18, 5]	6
1	3	1	[]	[18, 3]	8
2	5	1	[]	[]	4
3	1	1	[2]	[8]	10
4	2	2	[3, 2, 1]	[3]	6
5	7	1	[]	[17]	10
6	1	1	[2]	[]	13
7	1	1	[2, 6]	[19, 4, 3]	4
8	1	1	[]	[14]	7
9	5	1	[2, 3]	[]	1
10	1	2	[5, 3]	[7, 6]	12
11	5	1	[9]	[]	1
12	4	2	[]	[]	6
13	7	1	[]	[18, 12]	11
14	7	1	[6, 2, 1]	[0, 15]	9
15	2	1	[0]	[14, 3]	4
16	2	1	[]	[2]	5
17	6	2	[]	[]	12
18	3	2	[0, 2]	[]	9
19	2	2	[13, 6, 15]	[]	2
20	2	1	[0, 5]	[19]	5
21	4	2	[18, 12]	[]	6
22	1	2	[]	[]	4
23	5	1	[0, 1]	[7]	9
24	6	2	[11]	[]	6

Procedure `CheckCompatibility`(job_j,ecu_i)
  $R = \{NULL\}$;
  $c = 1$ *if* $\forall j_k s.tD_j^k == 1, s_k == 2$ *otherwise* 0;
  if *c==1* **then**
    for *all* e_j *in* ECU_i *where* $(VM_i - length(JVM_{i,r})) - RVM_j >= 0$ **do**
      if $\nexists\ j_k \in e_j$ *suchthat* $c_i^k == 1$ **then**
        return *1*;
      end
    end
  end
  return *0*;

Procedure `F`(job_j,ecu_i)
  $fitval = CheckCompatibility\,(ecu_i, job_j) \times \frac{(\zeta \times (1 - NE_r) + \eta \times NRU_r)}{(\zeta + \eta)}$;
  return *fitval*;

5 Experimental results and discussion

In this section, numerical results are presented to evaluate the performance of the proposed CSJS algorithm. In order to validate the results, the proposed algorithm is simulated using the MATLAB simulator.[1] Here, it is assumed that the number of jobs=250, and number of ECUs=10. Number of VCs of each ECU arbitrarily varies from 1 to 7. Power consumption by active ECU, active VMs and inactive VMs are α, β, γ, respectively following the values stated in [23]. The code is also executed in Python 10.8.3 version for verification of the opted result.

In Table 4, the default parameter settings for evaluating CSJS are listed. The values are taken following the datasheet [23]. A case study is presented first to show the detailed working of the algorithm. For explaining the case study, a small set of 24 jobs has been considered to ease our explanation domain. However, for calculating the results, 250 jobs are considered. A brief discussion of the experimental results is presented in the subsequent subsections.

5.1 Case study

A case study has been performed with 25 jobs. In Table 5, a job is described by the following parameters.

- Id: Provides the unique identification number of the job.
- Priority: Priority of a particular job.
- Dependency: Provides the list of the job ids, on which the job is dependent.
- Conflict: Provides the list of the job ids with which the particular job has a conflict.
- Duration of a particular job w.r.t round.

Table 6 has provided optimized job schedule for the given job description in Table 5 as the output of the proposed CSJS algorithm. It has shown round wise details, i.e; on which round, which job will be placed on an ECU such that the resource utilization is maximized as well as the energy consumption is minimized. It is to be noted that in Table 6 some fields are left blank as no jobs are assigned to the VMs of the ECUs for that particular round. From Table 6 it is also seen that any new ECU is assigned only if no other active ECUs have sufficient number of VMs to execute a particular job. The resource utilization is calculated on the basis of the VMs used in the active ECUs. Even with heterogeneous ECUs with job conflicts and dependencies, the resource utilization is found to be around 82% which is quite appreciable.

From Tables 5 and 6, it can be observed that the algorithm is focusing on job priority. For example, job 8 and job 22 started their execution first, having priority 1, provided they do not have any job dependency, or job conflict with the jobs in available VMs.

[1] https://www.mathworks.com/products/matlab.html.

Table 6 Round wise execution details of a case study with jobs described in Table 5

Round	ECU1				ECU2		ECU3			ECU4				ECU5	Resource utilization
1	22	22	8	16	1	2	12	12	0	17	17	13		5	0.816666667
2	22	22	8	16	1	2	12	12	0	17	17	13		5	0.816666667
3	22	22	8	16	1	2	12	12	0	17	17	13		5	0.816666667
4	22	22	8	16	1	2	12	12	0	17	17	13		5	0.816666667
5	6		8	16	1	3	12	12	0	17	17	13		5	0.766666667
6	6		8		1	3	12	12	0	17	17	13		5	0.716666667
7	6	15	8		1	3	18	18		17	17	13		5	0.7
8	6	15			1	3	18	18		17	17	13		5	0.65
9	6	15	23			3	18	18		17	17	13		5	0.6
10	6	15	23			3	18	18		17	17	13		5	0.6
11	6	23	20			3	18	18		17	17	13			0.666666667
12	6	23	20			3	18	18		17	17				0.604166667
13	6	23	20			3	18	18							0.638888889
14	6	23	20			3	18	18							0.638888889
15	6	23	20	9	10	10	18	18		4	4				0.791666667
16	6	23	21	21	10	10	11			4	4				0.708333333
17	6	23	21	21	10	10	24	24		4	4				0.791666667
18	7	14	21	21	10	10	24	24		4	4	19	19		0.916666667
19	7	14	21	21	10	10	24	24		4	4	19	19		0.916666667
20	7	14	21	21	10	10	24	24		4	4				0.791666667
21	7	14	21	21	10	10	24	24							0.888888889
22		14			10	10	24	24							0.638888889
23		14			10	10									0.625
24		14			10	10									0.625
25		14			10	10									0.625
26		14			10	10									0.625

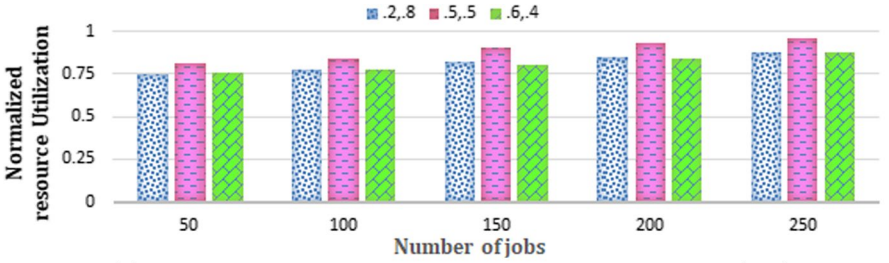
Handling of job dependency by the algorithm is reflected in Table 6. For example, job 6 has priority 1, but it is not able to start its execution till round 5 when job 2 finishes its execution, as job 6 is dependent on job 2 (shown in Table 5).

The algorithm also handles job conflict as is reflected in Table 6. For example, in round 5, job 3 is placed in ECU2, though there was sufficient VM available in ECU1 because job3 has a job conflict with job 8 (shown in Table 5).

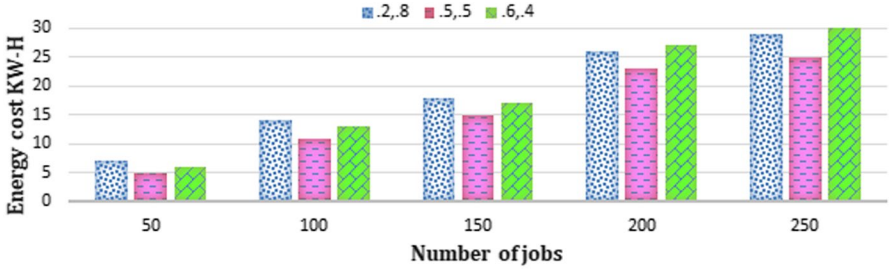
5.2 Performance tuning of the proposed CSJS algorithm

The default values of the setup parameters taken in this section are listed in Table 4. Thus, the results are reported for 250 jobs with 10 ECUs. It is important to find out the optimized value of the weight parameters ζ and η used in Algorithm 3 for calculating the optimized job schedule. In Fig. 2, the consumed energy cost and resource utilization with respect to different number of jobs are tested with varying the values of (ζ, η), respectively, where the weight coefficient of each evaluation factor satisfies $\zeta + \eta = 1$.

Different representative value combinations are explored in order to cover the entire horizon. As can be observed from the figures, the value combination $(\zeta = 0.5, \eta = 0.5)$ gives the best results. Thus, equal weightage for both the factors seem to work best for job selection as it maximizes resource utilization (Fig. 2a) while consumes less energy as shown in Fig. 2b. For rest of the experiments, both the factors are given equal weightage accordingly.



(a) Normalized resource utilization with varying constants (ζ, η)



(b) Energy cost KW-H with varying constants (ζ, η)

Fig. 2 Effect on different metrics with varying constants $(\zeta, \eta$

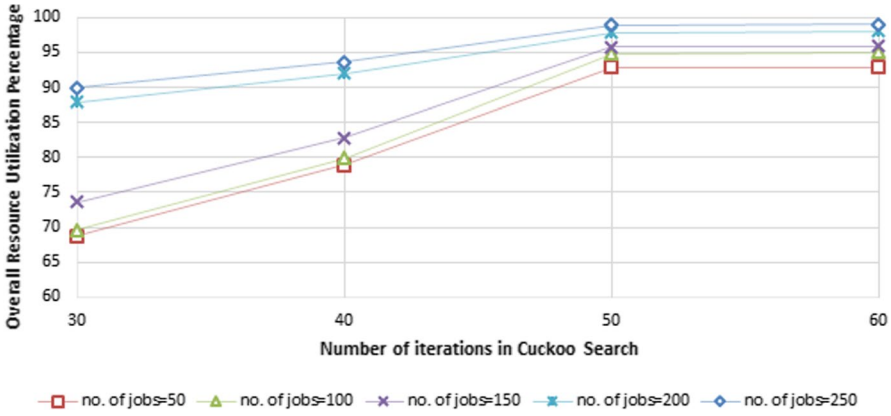


Fig. 3 Convergence performance of the proposed CSJS algorithm

In order to ensure reproducibility of the results it is important to ensure that the proposed algorithm converges after a finite number of iterations. Convergence of the CSJS algorithm is explored and the result is shown in Fig. 3. It is found that the outcome stabilizes after 50 iterations for varying number of jobs. Thus, the subsequent results are taken at more than 50 iterations in order to report stable output.

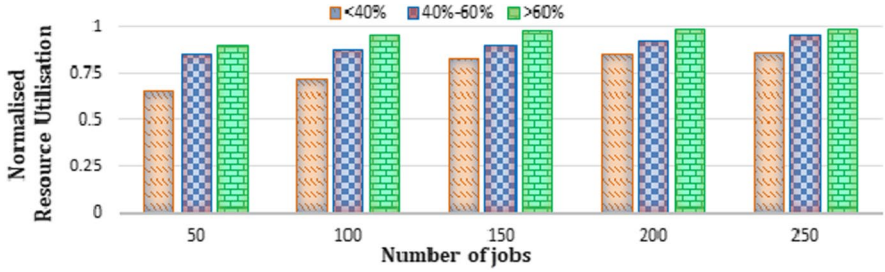
5.3 Analysis of VM requirements

It is important to investigate the required number of VMs based on the demand, that is, the jobs to be assigned for getting the optimized result. An experiment has been conducted, by varying the total number of VMs with respect to different number of jobs. Two different scenarios are considered as follows.

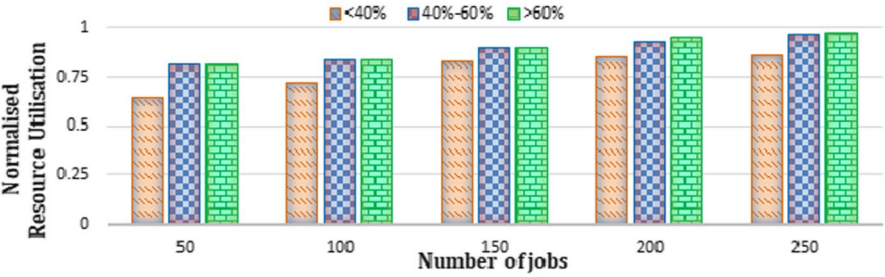
1. Without considering job conflicts and dependency and
2. Considering job conflicts and dependency.

Figure 4 shows the resultant output for the metrics resource utilization (Fig. 4a, 4b) and energy cost consumption (Fig. 4c, 4d), respectively. It is clearly shown in the figures that when there is no job conflict and dependency, with the increase of number of VMs, the resource utilization increases (Fig. 4a) and simultaneously, the consumption of energy is decreased (Fig. 4c). However, when job conflict and dependency are considered, the resource utilization gets increased up to 60% (shown in Fig. 4b) but beyond that, no further improvement could be observed because of job dependencies. Thus, even if the edge infrastructure is increased, it cannot be utilized beyond a point since the dependent jobs need to wait till their requisite jobs finish execution.

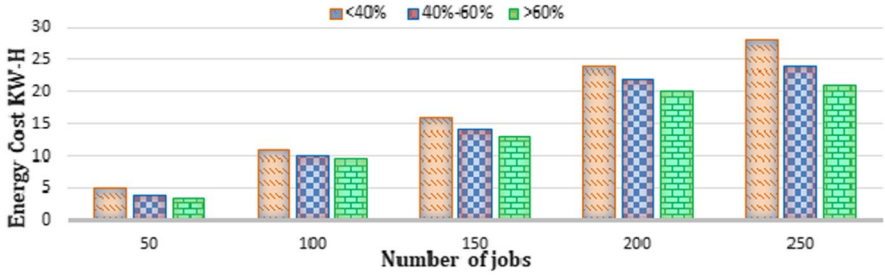
An experiment is conducted to further dig into the relationship between the ECUs along with its VMs and the number of jobs. Figure 5 shows the results of the experiments that has been conducted with even and uneven distribution of VMs in the ECUs. It is shown in the figure that even distribution, that is, homogeneous VMs



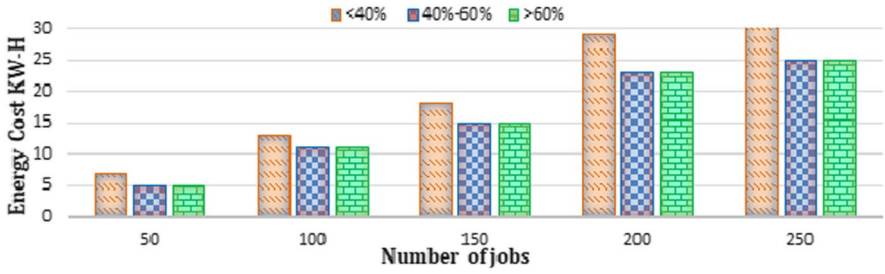
(a) Normalized resource utilization with no job conflict and dependency



(b) Normalized resource utilization with job conflict and dependency

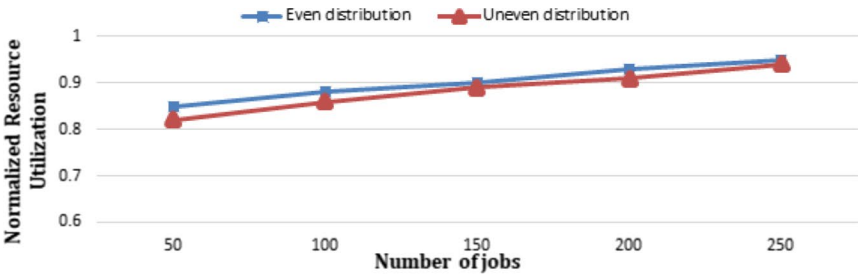


(c) Energy consumption with no job conflict and dependency

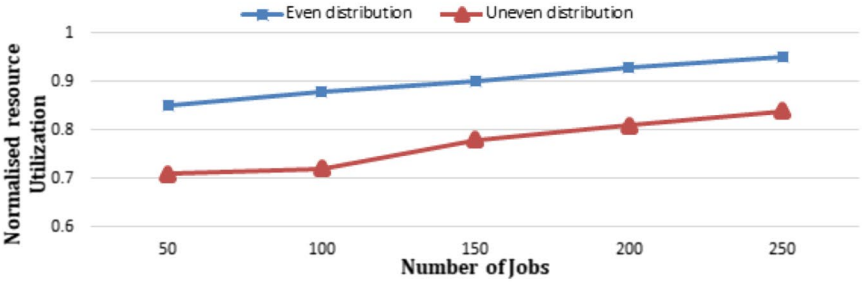


(d) Energy consumption with job conflict and dependency

Fig. 4 Variation of energy consumption and normalized resource utilization for varying percentage of VMs w.r.t number of jobs

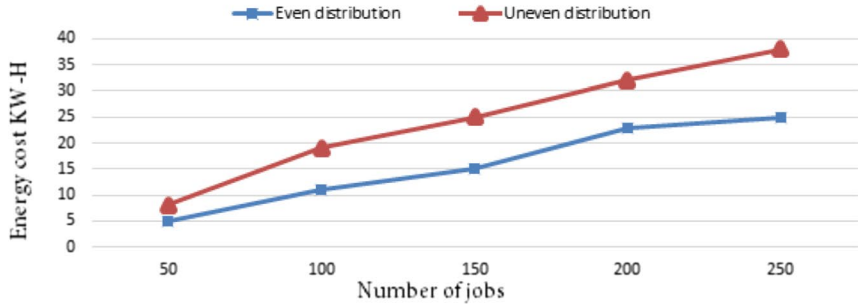


(a) Resource Utilization for homogeneous and heterogeneous ECUs with no job conflict and dependency



(b) Resource Utilization for homogeneous and heterogeneous ECUs with job conflict and dependency

(c) Energy consumption for homogeneous and heterogeneous ECUs with no job conflict and dependency



(d) Energy Consumption for homogeneous and heterogeneous ECUs with job conflict and dependency

Fig. 5 Variation of normalized resource utilization and energy consumption w.r.t even and uneven distribution of VMs in the ECUs

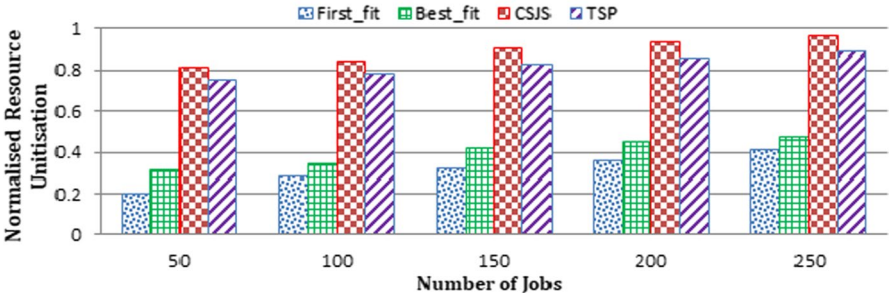
give better result in terms of minimum energy consumption and maximum resource utilization. Interestingly, it is also shown that, there is minor difference between the two scenarios when the job conflict and dependency are not considered (Fig. 5a, c). However, difference in performance could be observed when the job conflict and dependency are considered (Fig. 5b, d).

5.4 Comparison with state-of-the-art techniques

The experiments have been performed to compare the proposed algorithm with the state-of-the-art algorithms. For this purpose, the proposed algorithm has been compared with commonly used strategies and a state-of-the-art scheme TSP (Trust oriented IoT service placement) reported in [23]. The following commonly used intuitions are considered.

- BestFit where the jobs are placed in the ECU which has the least number of available VMs, but enough for execution of the job.
- FirstFit where jobs are placed to the first ECU having required number of VMS for execution of jobs.

(a) Normalized Resource Utilization without considering job conflicts and dependency



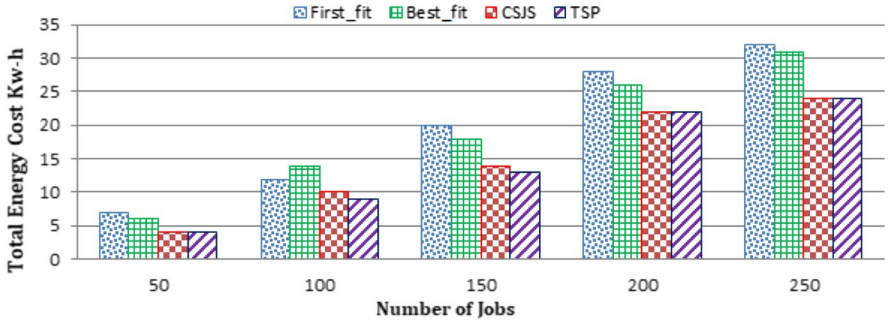
(b) Normalized Resource Utilization considering job conflicts and dependency

Fig. 6 Normalized resource utilization consumption w.r.t varying number of jobs

It is clearly seen from the Figs. 6a, b and 7a, b that, the proposed algorithm outperforms BestFit and FirstFit methods in terms of maximum resource utilization and minimum energy consumption. The proposed algorithm works in a comparable way with the work reported in [23] when no job conflict and dependency are considered and performs better than the work in [23] when job conflict and dependency are considered. This is because the proposed algorithm has considered dependency while populating the initial solutions for cuckoo search (reflected in line no 3 of Algorithm 2) and have considered job conflict while calculating the fitness value.

6 Conclusion

In this paper, we have introduced an optimal job scheduling algorithm for IoT in edge computing environment, based on cuckoo search algorithm. The algorithm is able to produce a suitable job schedule subject to different constraints such as, job priority, dependency among various jobs, and job conflicts. The algorithm ensures maximum resource utilization and minimum energy consumption. The algorithm supports heterogeneous ECUs consisting of varying number of VMs. The designed algorithm is implemented in MATLAB and is shown to perform



(a) Total energy cost consumption without considering Job conflicts and dependency

(b) Total energy cost consumption considering Job conflicts and dependency

Fig. 7 Total energy cost consumption w.r.t varying number of jobs

well in different conditions such as, with and without job conflict and dependency, and even and uneven distribution of VMs. The results show how the developed technique works better than a state-of-the-art technique in terms of energy consumption and resource utilization.

We would like to discuss certain limitation of our proposed method. First, the proposed algorithm hasn't considered the job migration from one ECU to another, as the job migration results in more energy consumption. Secondly, all the jobs are considered here to be non-pre-emptive. Thus, in the future, the pre-emptive job scheduling problem will be explored considering the situation where job migration is necessary.

Author Contributions MB: Conceptualization, Methodology, Investigation, Writing—original draft. CC: Methodology, Supervision, Writing—review and editing. UM: Writing—review and editing.

Funding The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

References

1. Skarlat O, Nardelli M, Schulte S, Borkowski M, Leitner P (2017) Optimized iot service placement in the fog. Serv Orient Comput Appl 11(4):427–443
2. Bakshi M, Chowdhury C, Maulik U (2021) Energy-efficient cluster head selection algorithm for iot using modified glow-worm swarm optimization. J Supercomput 77:6457–6475
3. Lin C-C, Deng D-J, Chih Y-L, Chiu H-T (2019) Smart manufacturing scheduling with edge computing using multiclass deep q network. IEEE Trans Ind Inform 15(7):4276–4284
4. Roy S, Chowdhury C (2017) Integration of internet of everything (ioe) with cloud. In: Batalla J, Mastorakis G, Mavromoustakis C, Pallis E (eds) Beyond the Internet of Things. Internet of Things (technology, Communications and Computing). Springer, Cham
5. Feng J, Liu Z, Celimuge W, Ji Y (2018) Mobile edge computing for the internet of vehicles: offloading framework and job scheduling. IEEE Veh Technol Mag 14(1):28–36
6. Dutta J, Roy S, Chowdhury C (2019) Unified framework for iot and smartphone based different smart city related applications. Microsyst Technol 25:83–96
7. Li C, Wang C, Luo Y (2020) An efficient scheduling optimization strategy for improving consistency maintenance in edge cloud environment. J Supercomput 66:1–28
8. Samanta A, Tang J (2020) Dyme: dynamic microservice scheduling in edge computing enabled iot. IEEE Internet Things J 7(7):6164–6174
9. Madej A, Wang N, Athanasopoulos N, Ranjan R, Varghese B (2020) Priority-based fair scheduling in edge computing. In: 2020 IEEE 4th International Conference on Fog and Edge Computing (ICFEC). IEEE, pp 39–48
10. Feng J, Liu Z, Celimuge W, Ji Y (2017) Ave: autonomous vehicular edge computing framework with aco-based scheduling. IEEE Trans Veh Technol 66(12):10660–10675
11. Han Z, Tan H, Li X-Y, Jiang SH-C, Li Y, Lau FCM (2019) Ondisc: online latency-sensitive job dispatching and scheduling in heterogeneous edge-clouds. IEEE/ACM Trans Netw 27(6):2472–2485
12. Varshney P, Simmhan Y (2020) Characterizing application scheduling on edge, fog, and cloud computing resources. Softw Pract Exp 50(5):558–595
13. Luo Q, Hu S, Li C, Li G, Shi W (2021) Resource scheduling in edge computing: a survey. IEEE Commun Surv Tutor 6:66

14. Huang J, Li S, Chen Y (2020) Revenue-optimal task scheduling and resource management for iot batch jobs in mobile edge computing. Peer-to-Peer Netw Appl 13(5):1776–1787
15. Fan G, Chen L, Huiqun Yu, Qi W (2020) Multi-objective optimization of container-based microservice scheduling in edge computing. Comput Sci Inf Syst 00:41–41
16. Jiang K, Ni H, Sun P, Han R (2019) An improved binary grey wolf optimizer for dependent task scheduling in edge computing. In: 2019 21st International Conference on Advanced Communication Technology (ICACT). IEEE, pp 182–186
17. Liu L, Tan H, Jiang SH-C, Han Z, Li X-Y, Huang H (2019) Dependent task placement and scheduling with function configuration in edge computing. In: 2019 IEEE/ACM 27th International Symposium on Quality of Service (IWQoS). IEEE, pp 1–10
18. Meng J, Tan H, Li X-Y, Han Z, Li B (2019) Online deadline-aware task dispatching and scheduling in edge computing. IEEE Trans Parallel Distrib Syst 31(6):1270–1286
19. Cai L, Wei X, Xing C, Zou X, Zhang G, Wang X (2021) Failure-resilient dag task scheduling in edge computing. Comput Netw 66:108–361
20. Wang J, Liu Y, Ren S, Wang C, Wang W (2021) Evolutionary game based real-time scheduling for energy-efficient distributed and flexible job shop. J Clean Prod 293:126093
21. Li M, Chen C, Huaqing W, Guan X, Shen X (2020) Age-of-information aware scheduling for edge-assisted industrial wireless networks. IEEE Trans Ind Inform 17(8):5562–5571
22. Ullah I, Youn HY (2020) Task classification and scheduling based on k-means clustering for edge computing. Wirel Pers Commun 113(4):2611–2624
23. Xiaolong X, Liu X, Zhanyang X, Dai F, Zhang X, Qi L (2020) Trust-oriented iot service placement for smart cities in edge computing. IEEE Internet Things J 7(5):4084–4091
24. Aral A, Ovatman T (2018) A decentralized replica placement algorithm for edge computing. IEEE Trans Netw Serv Manag 15(2):516–529
25. Nguyen DT, Pham C, Nguyen KK, Cheriet M (2019) Placement and chaining for run-time iot service deployment in edge-cloud. IEEE Trans Netw Serv Manag 17(1):459–472
26. Mohiuddin I, Almogren A (2019) Workload aware vm consolidation method in edge/cloud computing for iot applications. J Parallel Distrib Comput 123:204–214
27. Yousefpour A, Patil A, Ishigaki G, Kim I, Wang X, Cankaya HC, Zhang Q, Xie W, Jue JP (2019) Fogplan: a lightweight qos-aware dynamic fog service provisioning framework. IEEE Internet Things J 6(3):5080–5096
28. Natesha BV, Guddeti RMR (2021) Adopting elitism-based genetic algorithm for minimizing multi-objective problems of iot service placement in fog computing environment. J Netw Comput Appl 178:102972
29. Ouyang T, Zhou Z, Chen X (2018) Follow me at the edge: mobility-aware dynamic service placement for mobile edge computing. IEEE J Sel Areas Commun 36(10):2333–2345
30. Sonkoly B, Haja D, Németh B, Szalay M, Czentye J, Szabó R, Ullah R, Kim B-S, Toka L (2020) Scalable edge cloud platforms for iot services. J Netw Comput Appl 170:102785
31. Choi J, Ahn S (2019) Scalable service placement in the fog computing environment for the iot-based smart city. J Inf Process Syst 15(2):440–448
32. Mseddi A, Jaafar W, Elbiaze H, Ajib W (2019) Joint container placement and task provisioning in dynamic fog computing. IEEE Internet Things J 6(6):10028–10040
33. Azizi S, Shojafar M, Abawajy J, Buyya R (2022) Deadline-aware and energy-efficient iot task scheduling in fog computing systems: a semi-greedy approach. J Netw Comput Appl 201:103333
34. Swarup S, Shakshuki EM, Yasar A (2021) Energy efficient task scheduling in fog environment using deep reinforcement learning approach. Procedia Comput Sci 191:65–75
35. Kumar MS, Karri GR (2023) Eeoa: cost and energy efficient task scheduling in a cloud-fog framework. Sensors 23(5):2445
36. Khaleel MI (2023) Efficient job scheduling paradigm based on hybrid sparrow search algorithm and differential evolution optimization for heterogeneous cloud computing platforms. Internet Things 66:100697
37. Yang X-S, Deb S (2010) Engineering optimisation by cuckoo search. Int J Math Model Numer Optim 1(4):330–343
38. Ghosh TK, Das S, Barman S, Goswami R (2017) A comparison between genetic algorithm and cuckoo search algorithm to minimize the makespan for grid job scheduling. In: Sahana SK, Saha SK (eds) Advances in Computational Intelligence, Singapore, 2017. Springer, Singapore, pp 141–147
39. Yang X-S, Karamanoglu M, Ting TO, Zhao Y-X (2014) Applications and analysis of bio-inspired eagle strategy for engineering optimization. Neural Comput Appl 25(2):411–420

40. Yang X-S, Karamanoglu M (2013) Swarm intelligence and bio-inspired computation: an overview. Swarm Intell Bio-inspir Comput 66:3–23

COMPLIANCE REPORT TO THE REVIEWER'S COMMENTS

Thesis Title: **Resource Optimization Techniques for IoT-Edge Environment**

Submitted By: **Mohana Bakshi**

IndexNo. **284/18/E**

Reviewer #1 Comment:

Some formatting problems are present e.g. at page 84

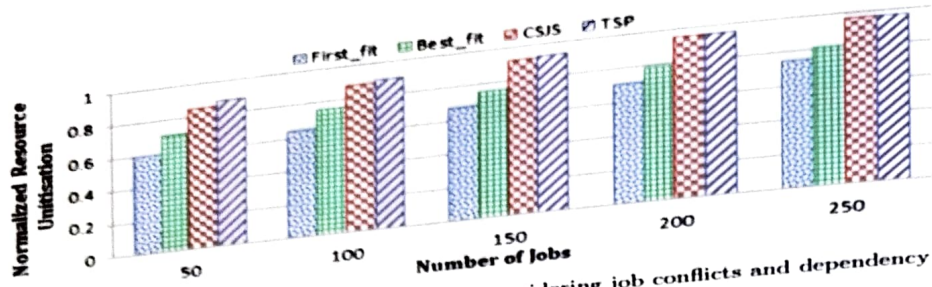
It is clearly seen from the figures Fig. 5.5(Fig. 5.5a, Fig. 5.5b) and Fig. 5.6(Fig. 5.6a, Fig. 5.6b). Please rectify those.

Response:

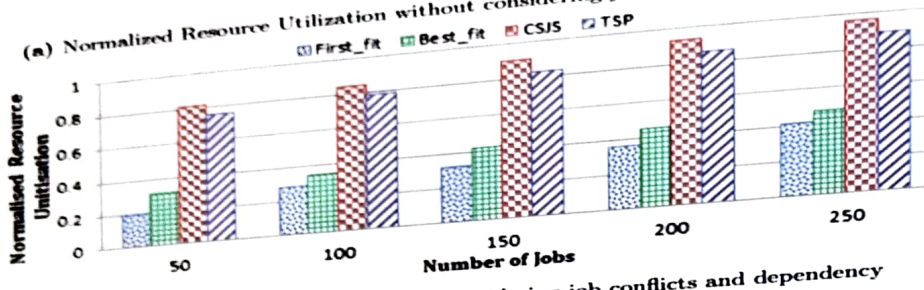
I would like to express my sincere gratitude to the respected reviewer for his valuable time and comments during the adjudication of my thesis.

In response to the observations regarding formatting issues—particularly with the figures on page 82 and page 83 (i.e., **Fig. 5.5 (a, b)** and **Fig. 5.6 (a, b)**)—the following corrections have been made:

- The **alignment of the figures** has been corrected to ensure visual coherence and proper layout within the document.
- Additionally, the **resolution of the figures** has been enhanced, thereby improving the accuracy and interpretability of the presented results.

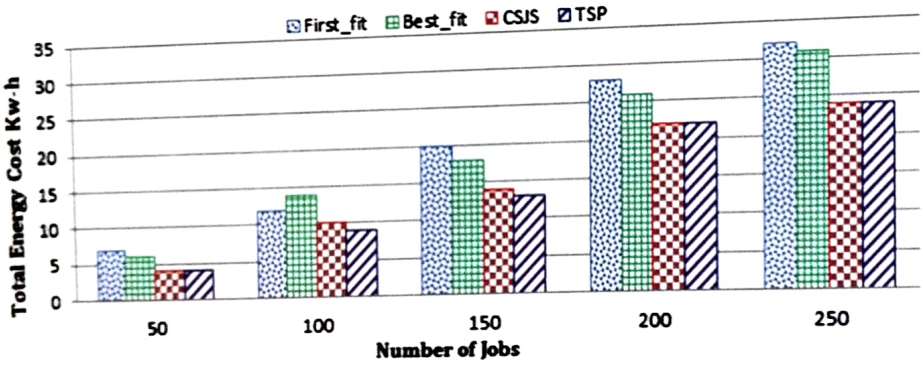


(a) Normalized Resource Utilization without considering job conflicts and dependency

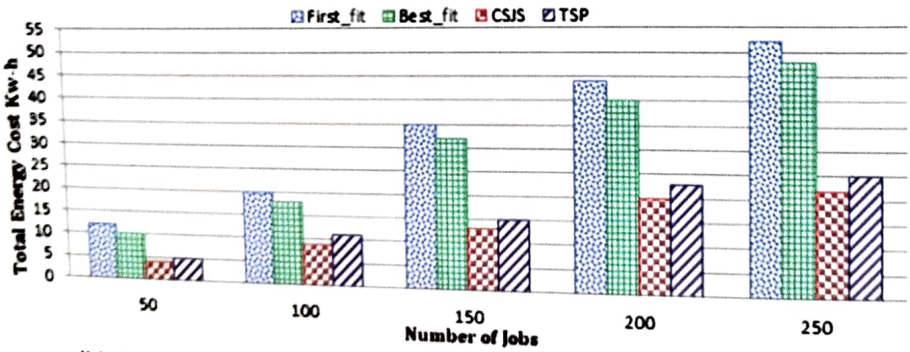


(b) Normalized Resource Utilization considering job conflicts and dependency

Figure 5.5: Normalized Resource Utilization consumption w.r.t varying number of jobs



(a) Total energy cost consumption without considering Job conflicts and dependency



(b) Total energy cost consumption considering Job conflicts and dependency

Figure 5.6: Total energy cost consumption w.r.t varying number of jobs

These revisions were made in accordance with standard academic formatting guidelines and with the goal of strengthening the overall quality and presentation of the thesis.

Reviewer #2 Comment:

The contributions adequately satisfy the requirements for the degree of Doctor of Philosophy in Engineering. The thesis is accepted for PhD (Engg.) degree.

Response :

I would like to express my heartfelt gratitude to the respected reviewer for the kind and generous acceptance of the thesis without any required changes.

The positive evaluation is deeply appreciated and serves as a meaningful affirmation of the work undertaken.

Mohana Bakshi
11-08-2025