Rani et al. LGURJCSIT 2023

ISSN: 2521-0122 (Online)
ISSN: 2519-7991 (Print)

LGU Research Journal of Computer Science & IT

doi: 10.54692/lgurjcsit.2023.074486

Vol (7): Issue (4), October – December 2023

Multi-Objective Optimization Based Distributed Task Offloading in Fog Computing

Tehmina Rani[1], Bushra Jamil[1*], Humaira Ijaz[1]
Department of CS & IT, University of Sargodha, Sargodha, Pakistan.

Email: bushra.jamil@uos.edu.pk

ABSTRACT:

The Internet of Things, with its promise of ubiquitous connectivity, leads to the connectivity of billions of devices continuously generating a sheer volume of data processed by Cloud-centric Internet of Things (CIoT) based architecture. Cloud data centers reside multi-hop away from the end user, resulting in certain limitations like long latency, bandwidth, and scalability. Fog computing addresses these challenges by extending cloud-computing capabilities to the edge of the network, thus alleviating these concerns. However, the efficient execution of diverse nature fog applications on these distributed, heterogeneous, and resource-constrained fog devices needs efficient resource management techniques. Among these techniques, distributed task offloading is the one that efficiently moves the tasks from resource-limited fog devices to multiple resource-rich devices. An efficient distributed task-offloading algorithm is imperative to minimize latency and cost, optimize resource utilization, conserve bandwidth, and improve the quality of service. In this paper, we propose a multi-objective optimization-based distributed task-offloading algorithm based on NSGA-II, which reduces latency, network utilization, cost, and energy consumption. We evaluate the proposed offloading algorithm using iFogSim in comparison with PSO and FCFS approaches. The results show that the delay, cost and network usage of the proposed algorithm is much better as compared to baseline algorithms.

KEYWORDS: Fog Computing, CIoT, Multi-Objective Optimization, Distributed Task Offloading.

1. INTRODUCTION

IoT is an environment that consists of a large no of heterogeneous interrelated devices over the internet for connecting and sharing data with other devices without human interference [1]. With the advancement of communication technology, the number of IoT and connected devices has increased, which produces an enormous amount of data. Initially, Cloud-centric Internet of Things (CIoT) based architecture is used, but it is difficult for the CIoT alone to handle this vast amount of data [2]. Furthermore, Cloud data centers are multiple hops away from the source of data origin. Transmitting data to the cloud will consume increased bandwidth and elevate latency and cost that's why cloud computing is not an ideal choice for real-time applications.

1.1. Fog Computing

To overcome these problems the concept of fog computing was introduced by Cisco, which extends cloud computing by moving the computing, processing, storage, and networking facilities to the edge of the network. Fog computing is a distributed computing infrastructure that provides low latency, efficient resource management, and real-time processing near the edge of the network [3]. The hierarchical, bi-directional and distributed architecture of fog computing is shown in Figure 1.

A. Edge Tier

The devices in the Edge tier act as the entry points to the fog computing net-work such as IoT sensors, smartphones, cam-eras, and embedded systems. These devices gather data and perform

initial data pro-cessing near the data source.

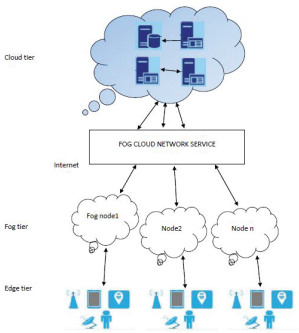

Figure 1: Fog Computing Architecture

B. Fog Tier

Fog tier consists of intermediary computing devices situated closer to the edge devices than traditional cloud data centers like micro data centers, routers, and switches. These nodes collect data from the edge tier, process it, and perform necessary computa-tions for intelligent decisions.

C. Cloud Tier

The topmost tier of the architec-ture, known as the cloud, is made up of a number of high-capacity data centers for long-term decision-making.

Fog computing has different characteris-tics such as heterogeneity, resource and ener-gy limitation, context awareness, and mobili-ty. The ever-increasing number of IoT appli-cations and devices generate an enormous amount of data that needs processing on these fog devices.

However, the dynamic nature, heteroge-neity, and resource-constrained fog devices make resource management a critical issue in fog computing. Therefore, efficient pro-cessing of the data generated by heterogene-ous IoT applications with different QoS re-quirements needs efficient exploration and use of available fog nodes via task offloading.

1.2. Distributed Task Offloading in Fog

Task offloading is a procedure of moving computing some tasks or workloads from resource-limited to resource-rich devices. In task offloading, the tasks are sent to a single node, while in distributed task offloading tasks are dynamically distributed across multiple fog nodes. Distributed task offloading deals with the intelligent allocation of tasks to the most suitable and available computing resources within the fog network based on various factors, including the computational requirements of the task, the proximity of resources, current resource utilization, and network conditions [4,5]. This ensures efficient resource utilization, minimizes latency, and enhances the scalability and fault tolerance of fog computing systems. Task offloading in fog computing is still in its early stages despite the existence of prior proposals for various task-offloading techniques. To identify their shortcomings in supporting the optimization of latency, network utilization, and energy consumption, we have conducted a brief literature analysis on the relevant literature about the various task offloading methodologies currently used in fog computing. It is discovered that the task offloading techniques now in use either complete mono-or bijective jobs or offload duties to a single node. Most task-offloading techniques pay attention to latency problems and overlook other factors. Therefore, it is necessary to develop a multi-objective optimization-based distributed task offloading strategy to distribute work among several fog nodes while optimizing performance and resource usage. For the implementation of NSGAII, PSO, and FCFS for distributed task offloading, we used iFogsim.This research paper is concerned with the de-sign, implementation, and evaluation of a multi-objective optimization-based distribut-ed task offloading algorithm that efficiently offloads tasks on fog nodes according to the requirements. The following are the main contributions of the suggested work.

i. We investigate metrics optimized and limitations of the existing task offloading algorithms.

ii. We design and implement an NSGA-II-based multi-objective optimization algo-rithm for distributed task offloading in a fog-cloud computing environment using iFogSim.

iii. We determine the effectiveness of our proposed algorithm in terms of latency, energy consumption, cost, and network usage and compare the results with the FCFS and Particle Swarm Optimization (PSO) algorithm.

The remainder of the article is structured as follows. In Section II, we present detailed the literature review that is closely related to our work. A summary and research solution for distributed task offloading based on multi-objective optimization in fog computing are presented in Section III. The evaluation process and simulation model utilized in this paper are

described in Section IV. The implementation of NSGAII and PSO for distributed task offloading is presented, and the outcomes of these algorithms are contrasted with FCFS. In Section V, we explain the research's outcomes and provide recommendations for further study.

2. RELATED WORK

In this section we have reviewed centralized, distributed, dynamic, hierarchical, heuristic, GA, and reinforcement learning-based algorithms that have been proposed for optimization of task offloading for the different metrics such as: Performance Metrics, Resource Usage, Financial Costs, and Energy usage. Load balancing was employed by Fricker et al. [6] to start offloading the jobs inside the fog datacenters. The authors outlined a situation in which a request arrives at a datacenter that is overburdened and is routed to a datacenter next door that has an equal chance of getting the request. According to their method, more requests are declined or banned depending on whether they can offload duties once a datacenter is overcrowded. The authors essentially offloaded work based on request blockage rate. Energy usage was emphasized by Zhang et al. [7]. They provided a mechanism for offloading computing in 5G networks that is energy-efficient. Their suggested offloading mechanism accounts for both the energy used during task execution and the energy used during transmission or transferring of the job. In order to reduce the total power use of the offloading process, the offloading system improves both the job offloading and radio resource allocation in 5G networks. Yousefpour et al. [8] delay-minimizing offloading strategy for fog nodes takes into account different request types with varying processing durations in addition to the queue's length. Following this, it decides whether or not to offload the chosen tasks to its best neighbor fog node if the projected waiting time of the fog node is more than an acceptable threshold.

Without using any categories, Aazam et al. [9] described the typical techniques utilized for work offloading and evaluation of previously published publications. The comprehensive literature review based on the choice of fog nodes throughout task offloading in VFC was reported by Hamdi et al. [10]. They concluded their work by discussing the problems and restrictions of their research. For single type offloading, Xu et al. Apply reinforcement learning techniques to lower overall cost in delay-sensitive applications [11]. A deep neural network (DNN) model that uses reinforcement learning to offload challenging tasks. Wang and fellows present a novel approach for task offloading for the edge that combines deep sequential models and reinforcement learning to optimize resource consumption in [12].

In order to solve minimize delay problem for task offloading in hierarchical fog computing network Pan et al., formulate the task offloading problem as linear integer problem [13]. Liu and fellows [14] delve into critical aspects of task offloading to meet the demands of ultra-reliable low latency application using extreme value theory. The primary goal of the study is to minimize the user's power consumption along with efficient resource allocation. By utilizing matching theory, Tran et al., investigated several offloading strategies for resource management [15]. In the NSGA-III algorithm, which is based on GA-based offloading, A queueing theory based task-offloading algorithm is suggested to minimize execution delay and optimize energy consumption in [16]. Nan and fellows propose an online algorithm based on Lyapunov optimization technique to balance the tradeoff between average response time, cost and number of application loss [17]. Xu et al., presented a multi-objective based computation offloading method (MOC) that has better latency and cost, but more work needs to be done to efficiently distribute resources for the Internet of Vehicles (IOV), such as delay management and energy consumption [18]. The authors claimed that their algorithm is not only cost effective but also minimize response time and average no of applications. Adhikri and fellows propose an innovative task offloading approach that combines deadline and priority requirements of tasks using multi-level feedback queues [19]. The algorithm offers minimal latency along prioritiaed execution. Chang et al., use game theory to optimize task offloading in multi-server edge computing in overlapping service areas of mobile users [20]. They apply non-cooperative game method using real-time update computation offloading (RUCO) algorithm that uses Nash equilibrium, and a multi-user probabilistic offloading decision algorithm to address this problem. A backpressure algorithm based task-offloading algorithm to minimize delay of latency-sensitive tasks in smart homes is presented in [21]. This algorithm minimize the queue length of tasks by minimizing Lyapunov drift optimization algorithm in each slot to improve the stability of

the queue and offloading performance. In 2021, Vu and fellows present a joint task offloading and resource allocation algorithm for multi-layer cooperative fog network that exploits energy efficient techniques ensuring to meet delay constraints and network performance optimization [22]. Hou and fellows propose a hierarchical task offloading technique for latency-sensitive and delay-tolerant applications that integrate artificial intelligence and edge computing to ensure the quality of service, minimized latency and service that is more reliable in [23]. In 2022, Malik and fellows present a matching based parallel offloading technique for optimizing resource utilization and minimizing latency in IoT networks [24]. The authors generated preference profiles for different IoT nodes, for task-offloading decision due to which latency is reduced up to 52% for heavy task load.Kishor et al. takes inspiration from nature to optimize task distribution in fog computing environments [25]. This study leverage smart Ant Colony Optimization (ACO) to increase the efficiency of resource allocation along with minimization of latency. Shi and fellows present a task-offloading algorithm based on deep deterministic policy gradient (DDPG) algorithm for vehicular fog computing environment [26]. The authors compared the results of proposed algorithm with deep Q-learning and actor-critic algorithm and claimed to get better results in terms of reducing cost. Tran-Dang et al. present a dynamic and collaborative approach for task offloading to process the data efficiently [27]. This study leverage the performance of fog computing by real-time collaboration between fog nodes to reduce average delay with high rate of service requests. Table 1, presents the comparison of the proposed task offloading techniques in terms of optimization metrics, and simulator used. Literature review and analysis show that existing task offloading approaches are either mono or bijective or offload tasks to one node. Most current algorithms focus on latency issues while ignoring other parameters during task offloading. Therefore, there is a need to design an efficient multi-objective optimization-based distributed task offloading algorithm that offloads tasks to multiple fog nodes along with efficient utilization of resources and performance optimization.

Table 1: Comparison of Existing Task Offloading Algorithms

Sr. #	Tech.	Metrics	Simulator	Ref.
1	Analytical Modelling	Latency Service delay	–	8
2	DRL	Latency	–	12
3	Majorization-minimization	Latency	Monti-Carlo	13
4	Extreme value theory	Delay Power con-sumption	–	14
5	Lyapunov optimization	Response time, cost, no of applica-tion loss	–	17
6	NSGA-III	Energy con-sumption, delay	Cloudsim	18
7	Priority-based	Waiting time, throughput, deadline	–	19
8	Non-cooperative game method	cost	–	20
9	Lyapunov optimization	delay	–	21
10	Branch and Bound, FFBD	Energy con-sumption, delay	–	22
11	Multi-agent DRL	delay	–	23
12	Matching-based	delay	–	24
13	Meta-Heuristic	Latency, QoS	MATLAB	25
14	DDPG	cost	Python	26
15	DCTO	Reduce delay, high service rate	–	27

3. DESIGN AND IMPLEMENTATION

Tome of the tasks to the cloud or fog because we need an additional entity that executes the task and returns the result immediately after execution to support a real-time application and to increase efficiency. Effective task offloading strategies are necessary to effectively optimize resources, accelerate response times, and boost the effectiveness of complicated systems. In order to distribute tasks to various fog nodes while maximizing resource utilization, minimizing latency, minimizing cost, energy consumption, and network usage., we will develop an effective multi-objective optimization-based distributed task offloading approach.

3.1. Application types in Fog Computing

Fog computing brings processing, storage, and networking resources closer to where data is

created and consumed by extending the ideas of cloud computing to the edge of the network. This enables diverse applications to handle data more quickly, with less latency, and with more effectiveness. Here are a few examples of application types that gain from fog computing:

A. IoT (Internet of Things) Devices

Fog computing is especially helpful for IoT devices that produce an enormous amount of data. You may decrease the amount of data that has to be transferred to the cloud, cutting latency and preserving bandwidth, by processing data locally on edge devices or in close-by fog nodes.

B. Smart Cities

Fog computing can be used for a number of smart city applications, including traffic, waste, and environmental monitoring. Local processing may make it possible to react to situations and occurrences more quickly.

C. Healthcare

In the field of healthcare, fog computing may be used for wearable health equipment, remote patient monitoring, and even real-time picture processing. This enables doctors to diagnose and decide more quickly.

D. Transportation

Intelligent transportation systems may greatly benefit from the use of fog computing. It can help with low-latency applications including real-time traffic monitoring, autonomous driving, and vehicle-to-vehicle communication.

3.2. Case study

Real-time data processing and analysis are often used in healthcare applications for patient monitoring and evaluation. This case study investigates the application of distributed task offloading using NSGA-II and PSO algorithms inside a fog computing architecture to solve the issues of latency, energy consumption, and resource efficiency in healthcare contexts.

A. Scenario

Wearable medical technology is used in hospitals to track patients' vital indicators including heart rate, blood pressure, and oxygen saturation. These gadgets continually gather information and send it to a centralized system for evaluation and diagnosis.

B. Challenges

Accurate results and quick responses are essential for real-time health data processing. However, analyzing such data on wearable devices with limited resources may be computationally and energetically taxing. It is crucial to offload work to adjacent fog nodes while maintaining low latency and efficient energy use.

C. Solution

A distributed task offloading algorithm is designed based on NSGA-II and PSO algorithms to handle the following issues:

- ***Issue Propagation***

i. Reduce Latency: The main objective is to reduce the amount of time needed to process and analyze health data in order to ensure prompt diagnosis and action.

ii. Reduce Energy Consumption: The second objective is to reduce energy use while carrying out tasks through wearable technology and fog nodes.

- ***Population Initialization***

Each potential solution in the population represents a method for job offloading. Each solution specifies, taking into consideration processing power and energy resources, which tasks are delegated to which fog nodes.

- ***NSGA-II and PSO Execution***

In order to identify a collection of Pareto-optimal solutions that balance minimizing delay and energy usage, NSGA-II develops the population. In order to identify the best work offloading arrangements, PSO refines solutions by simulating particle motion in the solution space.

- ***Fitness Evaluation***

On the basis of task execution time, communication latency, and energy consumption, solutions are assessed. The properties of wearable technology, fog nodes, and the communication network are used to determine these measures.

- ***Pareto Front Selection***

The Pareto-optimal solutions found by NSGA-II reflect various trade-offs between reducing latency and maximizing energy efficiency.

- ***PSO Swarm Update***

PSO particles modify their locations to move in the direction of the NSGA-II's best conclusions.

The task offloading configurations are improved by the swarm exploration method.

- ***Offloading Decision***

The trade-offs between minimizing latency and energy usage are taken into consideration while making final offloading selections. In order to save energy, fewer time-sensitive processes might be offloaded to the cloud while critical jobs may be offloaded to fog nodes for speedy analysis. Fog computing's distributed work offloading system enhances real-time health data analysis in healthcare settings by combining NSGA-II and PSO algorithms. In a hospital setting, this method assures prompt replies, resource efficiency, and energy effectiveness, all of which improve patient care and medical decision-making.

3.3. Proposed NSGA II Based Task Offloading Algorithm

We used the NSGA II algorithm to achieve the goal because it will reduce task latency or application loop delays by minimizing the average waiting time between tasks. Because the fog devices have limited resources, the fog systems have been adapted to produce tasks that are short and easy for these devices to handle. These algorithms distribute the tasks according on the MIPS. The NSGAII algorithm for distributed task offloading is given as:

Algorithm1: NSGAII algorithm for task offloading

```
Input: Tuples
While not end of tuples do
    Receive tuple
    If tuple arrive then
        Add tuple to the waiting list
        Not overloaded tuple
    Else
        Put it in the execution list
        Allocate vm to tuple
        Execute it
        Remove it from waiting list
    End
    If Overloaded tuple arrives then
        Add tuple to the waiting list
        Provide waiting list to NSGAII
        After calculations NSGAII return waiting
        list with best solutions
        Put the arranged tasks in execution list
        Allocate vm to tuple
        Execute it
        Remove it from execution list
        Add it to finish list
    End
End
```

A tuple is submitted to the scheduler whenever it arrives at the fog device. The task will be sent for execution if the scheduler determines that it is not overloaded; otherwise, it will be placed in the waiting queue that is kept at each fog device. The waiting queue is provided to the NSGA II algorithm and, after calculation, NSGA II returns the waiting list with the best solution and puts it into the execution queue. When a task has finished being executed, it is added to the finished queue, and the procedure is then continued.

The methodology of NSGA-II algorithm for distributed task offloading is shown in figure 2.

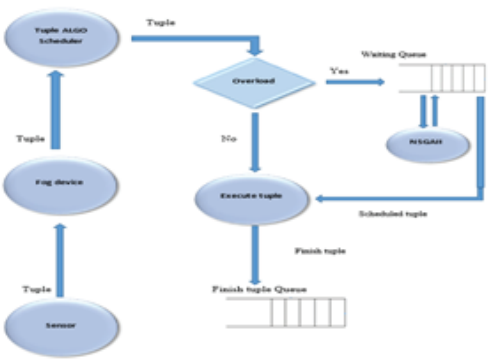

Figure 2: Methodology of NSGAII algorithm for Distributed task offloading

3.4. PSO Based Task Offloading Algorithm

We used the PSO algorithm to achieve the goal because it will reduce task latency or application loop delays by minimizing the average waiting time between tasks. Because the fog devices have limited resources, the fog systems have been adapted to produce tasks that are short and easy for these devices to handle. These algorithms distribute the tasks according on the MIPS.

Algorithm 2: PSO algorithm for task offloading

```
Input: Tuples
While not end of tuples do
    Receive tuple
    If tuple arrive then
        Add tuple to the waiting list
        Not overloaded tuple
    Else
        Put it in the execution list
        Allocate vm to tuple
        Execute it
        Remove it from waiting list
    End
    If Overloaded tuple arrives then
        Add tuple to the waiting list
        Provide waiting list to PSO
        After calculations PSO return waiting list with best
        solutions
        Put the arranged tasks in execution list
        Allocate vm to tuple
        Execute it
        Remove it from execution list
        Add it to finish list
    End
End
```

Algorithm 3 explains the functionality of PSO the algorithm. A tuple is submitted to the scheduler whenever it arrives at the fog device. The task will be sent for execution if the scheduler determines that it is not overloaded; otherwise, it will be placed in the waiting queue that is kept at each fog device. The waiting queue is provided to the PSO algorithm and, after calculation, PSO returns the waiting list with the best solution and puts it into the execution queue. When a task has finished being executed, it is added to the finished queue, and the procedure is then continued. Figure 3 presents the methodology of PSO algorithm for distributed task offloading.

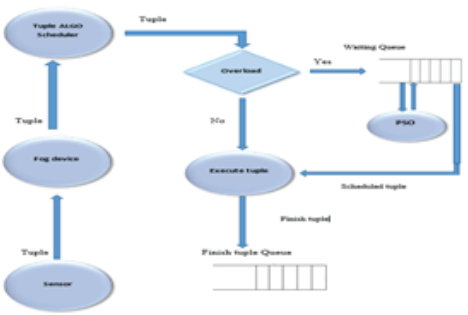

Figure 3: Methodology of PSO algorithm for distributed task offloading.

4. RESULTS AND DISCUSSIONS

4.1. Simulation setup

We have used iFogSim to simulate our offloading strategies. IFogsim is a powerful toolset for simulating resource management strategies in IoT and fog computing scenarios. To allow the deployment of distributed task offloading techniques for multi-objective optimization, we updated iFogSim. We add certain classes to be implemented over the scheduler, as well as update various iFogSim classes. Below is a quick overview of the classes that are commonly used:

A. Sensors

To simulate Internet of Things sensors, use this class. Tuples can be used to send data from sensor instances to Fog devices. This class is used to create tuples of different sizes.

B. Fog Device

This class's instances are used to represent various fog-generating devices. Memory, processing power, storage capacity, and uplink and downlink bandwidths are all included for each Fog device. Fog nodes may have numerous levels. The tuples are a means by which each fog node can interact with other fog nodes at a higher level and with objects inthe IoE layer. Each Fog node is capable of processing the arriving tuples that the scheduler has chosen based on MIPS.

C. Tuples

All of the fog's entities communicate with one another via instances of the tuple class. Each tuple is made up of source, destination, and processing demands expressed in MIPS.

D. Tuple Algo scheduler

This class is an extension of Cloudlet Scheduler, which manages three queues: the execution queue (QE), the finished queue (QF), and the waiting queue (Qw). All of the pairs on the waiting list are those that are awaiting execution, while those on the finish list have finished their execution. The NSGAII and PSO algorithms are implemented by the Tuple Algo Scheduler class by keeping the following three queues when a tuple is overloaded and delivered to the waiting list.

i. Cloudlet Exec List: This queue contains the cloudlets that are to be run on VMs.

ii. Cloudlet Finish List: This queue includes a list of cloudlets whose execution has been completed.

iii. Cloudlet Waiting List: These cloudlets are in this queue and are awaiting execution.

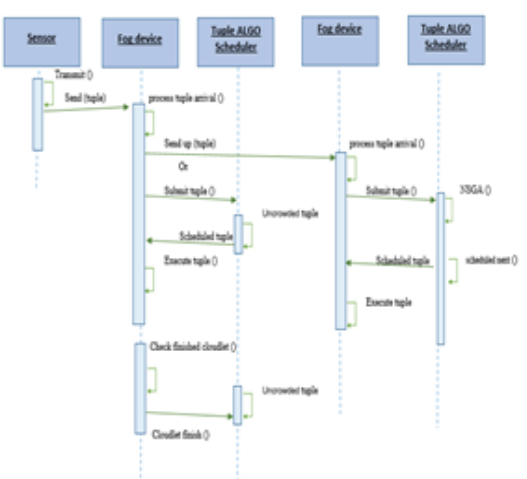

Figure 4: sequence diagram of NSGAII for distributed task offloading

Figure 4 displays the tuple emission, scheduling, and execution of NSGAII algorithm. In whicha tuple sent from a sensor using the Transmit() method is transmitted to a low-level connected

fog device using the Send (tuple) function. When a tuple arrives, the Fog device invokes the callback function processTupleArrival(). This method determines whether the tuple should be processed at the fog device or forwarded to a higher-level fog device. The SendupTuple() method sends the tuple to an upper level fog device if it is overloaded, otherwise the same level fog device processes it. Overloaded Tuples are sent to the TupleAlgoScheduler class, where tuples in the waiting queue QW are sent to the NSGA(). The load was distributed using NSGA() using the getsolution() technique. The scheduled tuple is then transmitted to the Fog device for execution once SchedulenextTuple() has chosen the next one. When a tuple is fully executed, the CloudletFinish () method sends the scheduler a request for the execution of the next tuple. The finished tuple is added to the finished tuple queue QF using this function.

Figure 5 displays the tuple emission, scheduling, and execution of PSO algorithm.

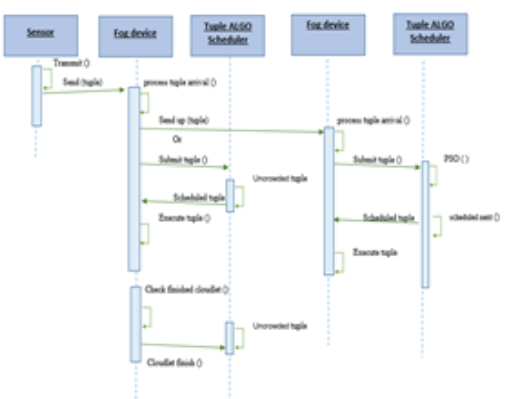

Figure 5: sequence diagram of PSO for distributed task offloading

As seen in the figure 5, a tuple sent from a sensor using the Transmit() method is transmitted to a low-level connected fog device using the Send(tuple) function. When a tuple arrives, the Fog device invokes the callback function processTupleArrival(). This method determines whether the tuple should be processed at the fog device or forwarded to a higher-level fog device. The SendupTuple() method sends the tuple to an upper level fog device if it is overloaded, otherwise the same level fog device processes it. Overloaded Tuples are sent to the TupleAlgoScheduler class, where tuples in the waiting queue QW are sent to the PSO(). The load was distributed using PSO() by execute() method. The scheduled tuple is then transmitted to the Fog device for execution once SchedulenextTuple() has chosen the next one. When a tuple is fully executed, the CloudletFinish() method sends the scheduler a request for the execution of the next tuple. The finished tuple is added to the finished tuple queue QF using this function.

4.2. Configurations

We have conducted thorough simulation-based research to investigate the effects of the NSGAII and PSO for distributed work offloading. In order to evaluate the effectiveness of these approaches, we ran a number of experiments using five overlay topologies with a total of 30, 35, 40, 45, and 50 nodes, respectively. These topologies combine nodes into four tiers, simulating the architecture of fog computing. The lowest layer is made up of sensors and actuators; the highest layer is made up of low-level fog devices; the second highest layer is made up of high-level fog devices; and the topmost layer is made up of clouds. We used high-intensity fog devices ranging from 1 to 5 for each set of experiments. The number of low-level fog nodes ranges from 6 to 10, accordingly. Each fog device receives information from sensors that are attached to it and takes appropriate action. For each configuration, the simulation takes 400 units of time. Table 2 displays the parameters for fog nodes at each level.

Table 2: Configuration of Fog nodes

Name	MIPS	RAM	UPBW	DNBW	Level
Cloud	3000	20000	100	5000	0
Base Station	2000	1000	5000	5000	1
2nd Level Fog Device	1000	10000	1000	5000	2
1st Level Fog Device	1000	5000	2000	5000	3

4.3. Distributed task offloading on fog devices

This section analyses the outcomes of distributed task offloading for the various types of load employed by various fog devices using FCFS, NSGAII, and PSO. The load used by all fog devices is indicated along the y-axis, while various devices are displayed along the x-axis.

Grey denotes the PSO, orange the NSGAII, and blue the FCFS. The FCFS result has varied loads on various devices, as shown in figure 6, however after using the suggested algorithms (NSGAII and PSO), all devices have almost identical loads.

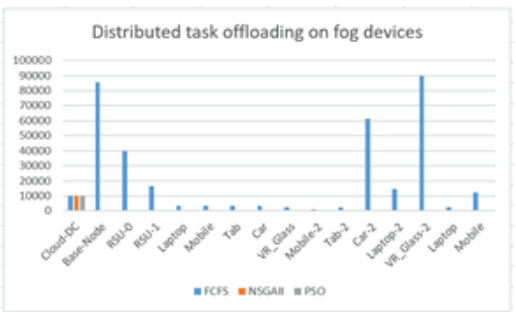

Figure 6: Distributed task offloading on fog devices

4.4. Performance metrics

We have chosen four criteria, namely loop delays; energy consumption; network utilization; and cost of execution to compare the performance of NSGAII and PSO for distributed task offloading against the FCFS algorithm.

A. Average Loop Delay

To evaluate the end-to-end latency of each module in the loop, we apply a control loop. We calculate the average CPU time, Tcpu, used by all tuples of a specific type in order to determine the loop delay. We use Equation 1 and 2 to calculate this average as given below.

$$T_{cpu} = \begin{cases} \frac{BT_i \times N + FT_i - BT_i}{N+1} & (1) \end{cases}$$

If the computed average CPU time for a specific type of tuple otherwise

$$FT_i - BT_i \qquad (2)$$

Where BTi is the beginning execution time by all tuples of a specific type of tuple, FTi is the finish execution time of ith tuple, and N is the total number of executed tuples of a specific type. We calculate the execution delay of every tuple by using this Equation (3).

$$Delay_i = BT_i - FT_i \qquad \forall i \in T \qquad (3)$$

Where T represents the current tuple set. Figure 7 displays the loop latency in milliseconds for various IoE node sizes. The PSO and NSGAII algorithms were used to compute it, and the results were compared to FCFS.

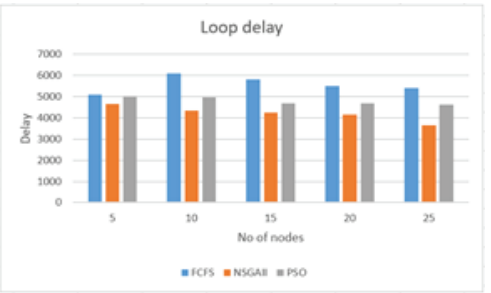

Figure 7: Application loop delay by applying PSO, NSGAII and FCFS

The x-axis shows the number of nodes, while the y-axis shows the latencies of the application loops. The average loop delay while using FCFS is depicted in blue, whereas the average loop delay when using NSGAII is depicted in orange, and the average loop delay when using PSO is depicted in grey. In contrast to NSGAII and PSO, the graph shows that FCFS has increased latency as the number of nodes rises.

B. Tuple CPU execution delay

This metrics defines the amount of time it takes to complete the processing of every type of tuple.

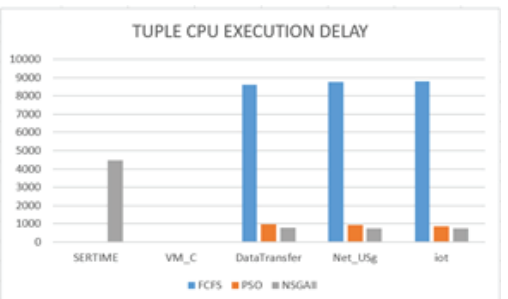

Figure 8: Tuple CPU execution delay of FCFS, NSGAII and PSO

Figure 8 displays the execution latency for tuples on the CPU. The PSO and NSGAII algorithms were used to compute it, and the results were compared to FCFS. The x-axis shows SERTIME, VM_C, data transmission, Net_usg, and IoT while the y-axis shows the application loop latencies. The average loop delay while using FCFS is depicted in blue, whereas the average loop delay when using NSGAII is depicted in orange, and the average loop delay when using PSO is depicted in grey. The picture shows that NSGAII has a greater SERTIME than the other two algorithms, although VM_C and are the same

for all algorithms. In contrast to NSGAII and PSO, Net_usg, data transfer and IoT of FCFS have longer tuple CPU execution delays.

C. Energy consumption

We calculate how much energy a Fog device uses E FN, by using Equation 4.

$$E_{FN} = E_p + (T_p - T_l) \times P_H \qquad (4)$$

The power of all the hosts within a specified time period can be used to calculate the energy of any Fog device, where Ep denotes the present energy consumption, Tp is the present time, Tl the update time of the previous utilization, and PH the host power during the last utilization.

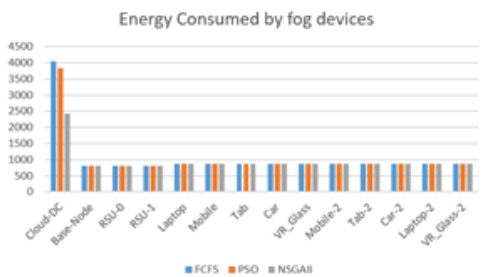

Figure 9: Average energy consumed by applying PSO, NSGAII and FCFS

Figure 9 shows the typical energy consumption of fog devices using the FCFS, NSGAII and PSO algorithms. The x-axis is used to indicate the devices, while the y-axis, or mega joules, is used to show how much energy each fog device uses. Orange represent PSO and blue reflect the FCFS findings, while grey displays the NSGA results. This graph illustrates the average energy consumption of algorithms and shows that NSGA consumes less energy than FCFS and PSO.

D. Network Usage

Network utilization Nu is the third evaluation parameter. As the number of devices grows, so does network usage, which causes congestion. Equation (5) is used to compute network usage for us.

$$N_u = \sum_{i=1}^{N} L_i * N_i \qquad (5)$$

Where Ni is the network size of the Ith tuple, Li is the latency, and N is the total number of tuples. Figure 10, shows how fog devices use the network and compares the FCFS algorithm to the NSGAII, PSO.

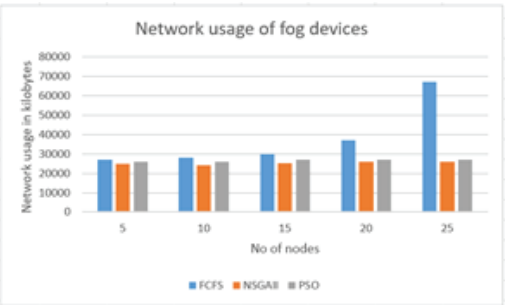

Figure 10: Network usage by applying NSGAII, PSO and FCFS

The x-axis shows the number of nodes, while the y-axis, or kilobytes, and displays the network size consumed by all fog devices. While blue indicates the FCFS, grey depicts the PSO findings, and orange displays the NSGA results. Figure 10 compares the network utilization of the NSGAII, PSO, and FCFS algorithms and demonstrates that, as the number of fog nodes rises, NSGAII and PSO utilize less network than FCFS.

E. Cost of Execution

One of the parameters used to evaluate the recommended module's reliability and accessibility is its execution cost. Execution cost can be compute by using Equation 6.

$$C_E = F_C + V_C / NUP \qquad (6)$$

CE represents "total execution cost," FC for "fixed cost," VC for "variable cost," and NUP for "number of units generated."

Figure 11 illustrates the execution cost required by fog devices and compares the FCFS method to the NSGAII algorithm and the PSO algorithm. The x-axis shows the number of nodes, while the y-axis displays the total cost of all fog devices. Grey depicts the PSO findings, orange the NSGAII results, and blue the FCFS results.

Figure 11: Cost of execution by applying PSO, NSGAII and FCFS

The outcomes demonstrate that the NSGAII performs better than PSO and FCFS for latency sensitive applications. The loop delay, energy consumption and network usage reduced by NSGAII as compare to PSO and FCFS. However, in cost of execution when the number of nodes increases, PSO cost of execution is lower than that of NSGAII and FCFS algorithms.

5. CONCLUSIONS AND FUTURE WORK

Due to an increasing number of IoT devices, a massive amount of data is generated daily. Cloud-centric Internet of Things (CIoT) based architecture is used for data processing, storage, and analysis. However, it is difficult for the CIoT to handle the sheer amount of data produced by these devices. To overcome these problems, the concept of fog computing is introduced that extends cloud computing by moving the computing, processing, storage, and networking facilities to the edge of the network. Due to resource-constrained edge devices, task offloading becomes a significant issue that needs attention. approaches for efficient and online-distributed task offloading. We shall use analytical modelling for capacity planning of these resource-constrained devices.In this paper, we have designed and implemented an NSGA II-based optimal distributed task offloading technique that is applied to a single overloaded fog node and distributes its load over several nodes. The proposed algorithm minimizes average loop latency, network use, and energy consumption by offloading tasks to fog devices based on duration. In the future, we shall apply reinforcement learning-based and multi-agent-based DRL approaches for efficient and online-distributed task offloading. We shall use analytical modelling for capacity planning of these resource-constrained devices.

REFERENCES

[1] M. H. Miraz et al., "Internet of Nano-Things, Things and Everything: Future Growth Trends," *Future Internet,* vol. 10, no. 8, pp. 68, DOI: 10.3390/fi10080068, 2018.

[2] A. R. H. Hussein, "Internet of Things (IOT): Research Challenges and Future Applications," *(IJACSA) International Journal of Advanced Computer Science and Applications,* Vol. 10, No. 6, 2019.

[3] M. Ghobaei-Arani et al., "Resource Management Approaches in Fog Computing: a Comprehensive Review," J. *Grid Comput.,* vol. 18, no. 1, pp.1-42, 2020.

[4] M. Y. Akhlaqi and Z. B. M. Hanapi, "Task offloading paradigm in mobile edge computing-current issues, adopted approaches, and future directions," *Journal of Network and Computer Applications,* 212, pp. 103568, 2023.

[5] J. V. Morey and S. K. Addya, "Efficient Task Offloading in IoT-Fog Network," In *Proceedings of the 24th International Conference on Distributed Computing and Networking,* (pp. 288-289), (January, 2023).

[6] C. F. Liu et al, "Dynamic Task Offloading and Resource Allocation for Ultra-Reliable Low Latency Edge Computing," *IEEE Communication Survey,* vol. 20, pp. 416–464, 2020.

[7] C. Fricker et al., "Analysis of an offloading scheme for data centers in the framework of fog computing," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS),* 1(4), pp. 1-18, 2016.

[8] A. Yousefpour et al., "On reducing IoT service delay via fog offloading," *IEEE Internet of things Journal*, 5(2), pp. 998-1010, 2018.

[9] M. Aazam et al., "Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities," *Future Generation Computer Systems,* 87, pp. 278-289, 2018.

[10] A. M. A. Hamdi et al., "Task offloading in vehicular fog computing: State-of-the-art and open issues," *Future Generation Computer Systems*, 133, pp. 201-212, 2022.

[11] J. Xu and S. Ren, "Online learning for offloading and auto scaling in renewable-powered mobile edge computing," *In Global Communications Conference (GLOBECOM), pp. 1-6, IEEE,* 2016.

[12] J. Wang et al. "Computation Offloading in Multi-Access Edge Computing

Using a Deep Sequential Model Based on Reinforcement Learning," *IEEE Communications Magazine,* 57(5), pp. 64-69, 2019.

[13] Y. Pan et al., "Latency minimization for task offloading in hierarchical fog-computing C-RAN networks," *In ICC 2020-2020 IEEE International Conference on Communications (ICC),* (pp. 1-6), IEEE, 2020.

[14] C. F. Liu et al., "Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing," *IEEE Transactions on Communications,* Vol. 67, no. 6, pp. 4132-4150, 2019.

[15] H. Tran-Dang and D. S. Kim, "A survey on matching theory for distributed computation offloading in iot-fog-cloud systems: Perspectives and open issues," *IEEE Access,* Vol. 10, pp.118353-118369, 2022.

[16] Z. Chang et al., "Energy efficient optimization for computation offloading in fog computing system," *In GLOBECOM 2017-2017 IEEE Global Communications Conference,* (pp. 1-6), IEEE, (December, 2017).

[17] Y. Nan et al., "A dynamic tradeoff data processing framework for delay-sensitive applications in cloud of things systems," *Journal of Parallel and Distributed Computing,* 112, pp. 53-66, 2018.

[18] X. Xu et al., "Multi-objective computation offloading for internet of vehicles in cloud-edge computing," *Wireless Networks,* 26, pp. 1611-1629, 2020.

[19] M. Adhikari et al., "DPTO: A deadline and priority-aware task offloading in fog computing framework leveraging multilevel feedback queueing," *IEEE Internet of Things Journal,* 7(7), pp. 5773-5782, 2019.

[20] S. Chen et al., "Distributed task offloading game in multiserver mobile edge computing networks," *Complexity,* pp. 1-14, 2020.

[21] Y. Wang et al., "Latency-optimal computational offloading strategy for sensitive tasks in smart homes," *Sensors,* 21(7), 2347, 2021.

[22] T. T. Vu et al., "Optimal energy efficiency with delay constraints for multi-layer cooperative fog computing networks," *IEEE Transactions on Communications*, 69(6), pp. 3911-3929, 2021.

[23] W. Hou et al., "Multiagent deep reinforcement learning for task offloading and resource allocation in cybertwin-based networks," *IEEE Internet of Things Journal,* 8(22), pp. 16256-16268, 2021.

[24] U. M. Malik et al., "Efficient Matching-Based Parallel Task Offloading in IoT Networks," *Sensors,* 22(18), 6906, 2022.

[25] A. Kishor and C. Chakarbarty, "Task Offloading in Fog Computing for Using Smart Ant Colony Optimization," *Wireless Personal Communications,* 127(2), pp.1683-1704, 2022.

[26] W. Shi et al., "Task Offloading Decision-Making Algorithm for Vehicular Edge Computing: A Deep-Reinforcement-Learning-Based Approach," *Sensors,* 23(17), pp. 7595, 2023.

[27] H. Tran-Dang, and D. S. Kim, "Dynamic Collaborative Task Offloading in Fog Computing Systems," *In Cooperative and Distributed Intelligent Computation in Fog Computing: Concepts, Architectures, and Frameworks*, *Cham: Springer Nature Switzerland*, (pp. 83-100), 2023.