National College of Ireland

A Novel Adaptive Energy-Aware Differential Evolution Algorithm for Fog Computing Optimization

MSc Research Project
Cloud Computing

Prakhar Singhwal
Student ID: x23285435

School of Computing
National College of Ireland

Supervisor: Prof. Sean Heeney

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Prakhar Singhwal
Student ID:	x23285435
Programme:	Cloud Computing
Year:	2024-2025
Module:	MSc Research Project
Supervisor:	Prof . Sean Heeney
Submission Due Date:	11/08/2025
Project Title:	A Novel Adaptive Energy-Aware Differential Evolution Algorithm for Fog Computing Optimization
Word Count:	7733
Page Count:	20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Prakhar Singhwal
Date:	11th August 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not	☐

sufficient to keep a copy on computer.	

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

A Novel Adaptive Energy-Aware Differential Evolution Algorithm for Fog Computing Optimization

Prakhar Singhwal
x23285435

Abstract

The expansion in the Internet of Things (IOT) devices has made room for a phenomenal increase in the output produced and instantaneous processing capacity. Traditional distributed computing systems, though efficient, seldom overlook energy-efficiency crucial for present IOT systems. Fog computing serves as an intermediary between cloud data centers and edge devices and are hosted near the edge nodes to lessen the response time and increase near source application execution efficiency. Nevertheless, these in-between nodes need to be in vicinity of the edge devices as they have limited processing power, especially in terms of energy. The best performing models focus on many metrics such as CPU Utilization, resource utilization often omitting energy efficiency. To conquer this disparity, the research proposes a novel adaptive differential evolution (ADE) algorithm which adapts dynamically to changing fog environment. The focus is to achieve exceptional task scheduling in a fog environment to cut down energy consumption keeping in mind that the other metrics remain unchanged. The outlined approach will be backed by a simulation tool (iFogSim) and will be measured against existing static and meta-heuristic algorithms. The designed solution is benchmarked at 50,100 and then 200 fog devices to see its effectiveness. Through the simulation, it was seen that the developed solution consumed 8.83% less energy than current state of art methods when evaluated at an intense workload of 200 fog devices.

Keywords: **IOT systems**, **adaptive differential evolution** (**ADE**),**iFogSim**, **energy efficiency**

1 Introduction

The connection point of a device or a local area (LAN) to the internet has experienced an outburst in data generated due to aggressive rise in Internet of Things (IOT) ecosystem. Due to billions of connected devices in healthcare, transportation, smart homes, and industrial systems, the need for real-time data processing and low latency systems is in huge demand (Yousefpour et al., 2018; Rego et al., 2022). The conventional cloud computing architecture is very scalable and centralized but is inefficient in managing latency, network congestion, offloading tasks to more powerful systems (Jimenez et al., 2019; Botta et al., 2016) which poses many difficulties to latency sensitive applications like video surveillance, augmented reality and remote surgeries. Fog computing has been suggested as a middle ground between the cloud and edge devices as the above-mentioned limitations offers cloud capabilities to the network present at the edge, making processing, storage, decision-making capabilities near

IOT devices (Abbas et al., 2020) .Fog computing have a localized computation which saves the use of centralized data centers and causes a substantial reduction in end-to-end latency, network congestion, and energy (Kang et al., 2021). This architecture uses fog nodes which are normally located closer to the edge. Despite all these upsides, these nodes are usually limited in terms of computing capability and energy supply when compared to cloud servers (Akbarzadeh et al., 2021). As a result, the issue of task scheduling in fog environment becomes a primary concern where one needs to balance between performance and energy efficiency.

To fill this gap, many algorithms which can enhance an issue by continuously trying to improve the picked solution are being used. One of them is Differential Evolution (DE) which is a metaheuristic algorithm based on population. The algorithm is preferred because it is fault tolerant and optimizes more than one objective function in the domain of mathematics concurrently. This solves the problems specified by (Hussain et al., 2021; Abbas et al., 2020). This optimization can be further developed by the addition of DE algorithm because of its easy and simple structure. Nevertheless, the algorithm does not have dynamic tuning so its performance is compromized in a fog environment where system's configuration such as CPU load and energy level changes very rapidly (Sharma et al., 2021).

The above-mentioned concern can be resolved if an algorithm can dynamically tune the fog nodes in an IOT environment. The proposed algorithm will make small changes to the parameters of the DE algorithm so that they are able to adapt to changing workloads. This will not only boost system performance but also execution efficirncy. Furthermore, the issues mentioned in the papers (Djumanazarov et al., 2020; Jimenez et al., 2019) such as finding different ways to transfer load to cloud in a dynamic way while ensuring that tasks are completed efficiently will be addressed.

There are many tools which can be used to deploy the fog infrastructure both in homogeneous and heterogeneous ways. Amongst them the algorithm will be evaluated on ifogsim2 toolkit. Apart from the proposed algorithm, other state of the art algorithms such as DE(Differential Evolution), GA(Genetic Algorithm),PSO(particle swarm optimization) will be evaluated and compared with the proposed algorithm. Performance indicators that will be measured are the time taken to execute tasks, CPU usage, energy consumption, and performance of the system in different load conditions. This study can help to develop intelligent fog computing systems that are efficient in terms of performance and are environmentally friendly due to their focus on adaptive scheduling and energy awareness. Such solutions are particularly valuable in the era of increasing interest in green computing and the necessity of resilient infrastructures that can support complex and dynamic workloads of the IoT. Moreover, in contrast to numerous area-specific solutions which can be used in a specific industry or fixed architecture (eg IIoT or centralized clouds), the proposed ADE framework is expected to be general and adaptable to fog-based implementations. This research will address one of the most significant gap in fog computing which is to create a self-adaptive scheduling algorithm which aims to improve energy consumption when compared to pre-existing algorithms.

1.1 Research Question

According to the introduction section provided above the interest in research project is outlined by the restated research question which shows the path for the analysis and derivation of the required results.

How can an Adaptive Differential Evolution algorithm be designed to optimize task scheduling in fog computing environments by balancing energy efficiency and execution performance under dynamic workload conditions?

1.2 Objectives of the research

The investigation has inclined to accomplish the following goals while building the application.
1: To work upon an Adaptive Differential Evolution (ADE) algorithm.
2: Integrate the algorithm with fog computing architecture.
3: Perform simulation using iFogSim2 toolkit.
4: Optimize energy consumption and efficiency.
5: Compare performance of proposed algorithm with existing algorithms.

Post this section related work in the field of fog computing where authors worked on different scheduling algorithms will be discussed in brief. After this tools, methodology, algorithms will be described in detail in section 3. Succeeding that experimental results will be compared with state of the art algorithms to determine the efficiency. Finally, the research will be concluded with results and the prediction of potential future work that can be done to improve the study.

2 Related Work

Due to tremendous rise in the internet of things(IOT) devices in the last decade the need of computing closer to the source where the data is produced is of paramount importance because processing this data in the cloud could result in higher network usage.This led to introduction of fog layer which acts as an intermediary layer between the cloud and the edge.However, these fog devices have limited processing computing capabilities so developing an efficient algorithm for scheduling these fog nodes would result in less energy consumption.A lot of work has been done previously in the area of scheduling fog nodes.This section of the document will explore and critically analyze the relevant work done in the area of fog and edge computing.

2.1 Evolution of Fog Computing for Task Scheduling

The theory of fog computing has now become an important standard that can fill the gap between cloud and edge devices. It increases the area of cloud to allow computation,storage and processing closer to the data source which minimizes latency and congestion experienced in cloud system. This architectural change is very beneficial in IOT intensive applications like smart healthcare, smart cities, smart agriculture and smart homes. The study by Jimenez et al. (2019) and Yousefpour et al. (2018) point to the fact that cloud system as a whole

cannot support latency and responsiveness that a real-time IOT application need. Distributed architecture not only reduces round trip time, but also maintains energy efficiency at the node level.

To deep dive into the domain of fog resource management and reduce overloading on edge nodes Akbarzadeh et al. (2021) explained the introduction of adaptive scheduling to Industrial IOT applications. He explained that the applications are still bound to static parameters and often ignores heterogeneous settings. On the contrast, Rego et al. (2022) suggest a fog computing model that focus on energy efficiency but does not include any scheduling logic in the model. He talks regarding the infrastructure while ignoring the organizing logic. Altogether, if both these works are combined it would give a complete view of how smart,adaptive and dynamic algorithms can operate and handle dynamic workloads.

2.2 Meta Heuristic Algorithms in Fog and Edge Computing

The algorithms which can find,generate,tune and provide sufficiently good optimization are needed to manage dynamic and complex fog nodes in real time. To achieve this they must be able to search large spaces and come up with an optimal solution quickly. There are many meta-heuristics algorithms which can handle IOT/fog workloads. To name a few there are DE(Differential Evolution), PSO(Particle Swarm Optimization), Ant Colony Optimization(ACO). Among them DE is the most widely used algorithm because of its simple and robust nature. Additionally, Hussain et al. (2021) used an adaptive DE (ADE) algorithm to find optimal service placemnet in the fog network. His outcome showed that although their was increase in energy efficiency and throughput the algorithm parameters were semi-static and did not include global feedback mechanism to adjust the evolution process with respect to real-time performance measures.

Abbas et al. (2020) also considered the same issues and suggested a context-aware DE-based deployment algorithm of IoT services in fog layers. They had several criteria in their model, which were the consumption of energy, service delay, and cost. Although it showed significant improvement in efficiency, the research did not attend to the task-level scheduling granularity, instead it concentrated on the high-level service deployment. Going further, Goudarzi et al. (2022) hybridized DE and Sparrow Search Algorithm to develop a strong job scheduling mechanism in cloud data centers. Nevertheless, they were still cloud-oriented and not easily applicable to the fog environment where nodes are resource-limited and volatile.

Real-time flexibility and elegant task placement continues to be an issue in such models. The majority of DE implementations employ fixed crossover and mutation factors which limits their application in dynamic environments where the load of a system and levels of energy vary in unpredictable ways. This drives the necessity of a new ADE method, specifically designed to work on fog computing and integrate the parameter tuning based on feedback, which will maximize scheduling effectiveness in dynamic environments.

2.3 Predictive Scheduling and Context-Aware Offloading

Predictive scheduling methods are used to help maximize resource allocation by looking at prior demands or by historical contexts to predict future demands. Benhida et al. (2021) have suggested a predictive scheduler of scientific workflows on edge devices that applies static workload profiling in decision-making. Although they work well in stable situations, their model does not have the versatility that is necessary in highly dynamic IoT applications. Moreover, the scheduler lacks energy trade-offs, which is why it is less appropriate to use it in sustainable environments.

Jimenez et al. (2019) proposed a fog-based model that took real-time context into account including node location and energy levels to optimize offloading. They were innovative but did not have evolutionary learning mechanisms, and decisions were made on the basis of rule-based heuristics, less scalable and more difficult to generalize. These models highlight the increasing awareness of context-aware computing but are unable to provide performance in high load system where high connectivity and dynamic environment is installed.

Such an adaptive DE-based scheduler can potentially outperform the static or rule-based predictive models, because it can adapt the strategy in response to real-time performance measures such as latency, energy usage, and CPU load.

2.4 Latency Minimization in Fog Environments

The applications that require high responsiveness in the network edge are latency-sensitive applications like augmented reality, smart traffic, and emergency health monitoring. Many works have tried to minimize the round-trip time, through efficient task placement strategies. One of the first models that consider the latency reduction through the opportunistic fog offloading was proposed by Yousefpour et al. (2018). Nevertheless, they failed to incorporate energy measures in their work, which resulted in a less than optimal performance of battery-limited devices.

In their fog-centric framework, Rego et al. (2022) focused on service responsiveness, which implies a load spread across the neighboring nodes. Although they promised significant reductions in delays, they did not include adaptive scheduling techniques, and were more interested in network-level optimization rather than computation-level control. These papers show a general trend which aims to improve a single metric, usually latency, however other metrics such as energy efficiency and scalability are sacrificed. Our proposed ADE algorithm fills this gap by applying multi-objective optimization to minimize latency, and control the energy consumption at the same time, which provides a more balanced and scalable real-time fog application scheduling system.

2.5 Distributed Energy Management and Infrastructure-Level Optimization

Some architectural-level solutions have been suggested to optimize power on a large scale in fog environment as well. In this study, Kebande et al. (2022) proposed a distributed fog orchestration framework of an intelligent energy coordination. The system's wide perspective enables them to make macro-level energy savings but does not allow responding to tasks, which is crucial in microservice-driven IoT systems. On the same note, Shakarami et al. (2022) introduced a multi-level architecture of smart building environments. Although they were useful in layered control, their scheduling policy was fixed and therefore could not be applied in variable situations.

Distributed frameworks play an essential role in providing baseline energy savings and resilience of the systems, but without dynamic scheduling mechanisms, localized optimization potentials are not fully realized. The suggested ADE algorithm would supplement such structures, offering task-level flexibility and adaptive parameterization, filling the divide between the infrastructure-level control and application-level reactivity.

2.6 Research Niche and Contribution

Based on the reviewed studies, it can be concluded that current scheduling solutions are more likely to focus on latency reduction, energy efficiency, or context-awareness, but rarely on all the three. Although the conventional DE algorithm has strong optimization features, they are not suitable in dynamic environments such as fog computing due to their statistic nature. Hybrid models and context-aware systems are one or two-dimensional enhancements that are not usually globally optimized or flexible.

The lack of a complete solution that combines the multi-objective optimization, feedback-based adaptation, and energy-aware responsiveness creates a gap in a critical research. Our proposed Adaptive DE algorithm is an attempt to solve this by proposing a lightweight, scalable and intelligent scheduling mechanism. It is adaptive to the environmental variability, balances the trade-offs amid latency,energy consumption and provides real-time scheduling of various fog applications.

3 Research Methodology

Performing live testing on IOT and fog devices can be very much expensive and complex because different components of the infrastructure could cost a heavy amount if the setup required is scalable and heterogeneous.Additionally, it is very hard to reproduce exact workloads or failure cases multiple times in live setups. For instance, scaling up the configuration from 50 fog devices to 500 fog devices is not possible to achieve due to limitation in hardware.Furthermore, testing energy aware scheduling algorithms can cause devices to overheat or disconnect which could ultimately lead to data corruption.To overcome all these challenges the best solution is to leverage a simulation tool which not only offer real-time scenario implementation but can scale easily to any workload.This is where iFogSim comes into play. ifogSim is a java based simulation toolkit which focuses on modeling and simulating IOT/fog computing resources.It is built on top of CloudSim and is very scalable and can evaluate multiple metrices.It is an open source toolbox which has a very big community who keeps on fixing issues and maintaining a clean repository. This paper shows how it can be used to design an adaptive energy aware algorithm for IOT/fog nodes.

3.1 IfogSim2 – A survey

Simulators changed over time to aid the experimenter to conduct and enhance their proposition and architecture. One of the most famous configuration over the internet is iFogSim, which has helped investigators to conduct experiments on edge devices. This tool is useful in scheduling, offloading and orchestrating various real-time IOT ecosystem. The main advantage of using this toolkit is that it can easily quantify the performance of various scheduling algorithms based on different metrics such as energy consumption, network usage, resource utilization etc. The toolkit has a five layered structure with cloud at the topmost layer and the sensors at the bottom. All the layers with their importance will be discussed in the subsequent sections.

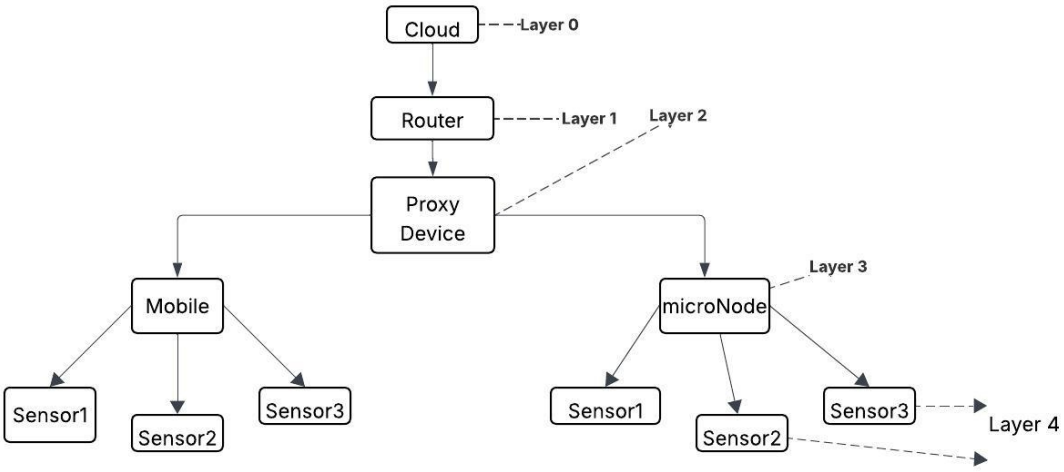

Fig 1: Layered Architecture of iFogSim-based Fog Environment

3.1.1 Components of iFogSim

The main parts of the iFogSim simulator are described below:

The Cloud Module: It is the top-level (Layer 0) of the fog computing hierarchy.It acts like a centralized data center. The purpose of this layer is to reproduce a public or private cloud infrastructure in a distributed IOT ecosystem.This element is responsible for storing huge data to process and access later in future**.**

The Router Module: The following level (Layer 1) sits between the cloud layer and the proxy layer. This layer is primarily responsible for connecting different networks in the system.In physical world they represent internet routers, switches etc.

The Proxy Module: This module is often referred to as the fog orchestrator as it helps to manage various fog/edge nodes. It performs different functions such as task scheduling, resource monitoring.This component receives the application module requests from sensors through edge/fog devices.Post this it evaluates a fog node using the scheduling algorithm and then deploys the task to the most efficient node.Overall, this module is responsible for task scheduling,resource allocation and decision-making.

The Mobile/microNode Module: The Layer 3 often represents devices which have less processing capabilities and are deployed closer to the data source so that data can be retrieved and processed in few-milliseconds giving almost negligible delay in response.These devices are usually smartphones,fitness bands which have less computing power and are present closer to the source just to cut-off the response time.

The Sensor Module: The bottom layer is the data generating layer which usually capture live physical/environmental conditions and convert them into digital data streams.For instance, any temperature conditions for a certain time period will be collected and sent as data streams to mobiledevices for storage.They reproduce the real-time behavior by periodically triggering the creation of application tuples.

3.2 Outline of Adaptive Differential Evolution (ADE) Algorithm

The Adaptive Differential Evolution algorithm is developed for live energy-control frameworks in area of modern IoT configuration using edge/fog computing. ADE boosts traditional Differential Evolution by announcing a combined adaptive mutation strategy which measure transformative parameters like mutation factor and crossover rate depending upon the evaluation of previous generation. Fog computing usually consists of distinct fog nodes and varying workloads with less processing capability. Conversely with the introduction of ADE all these limitations can be improved, as it is robust resource allocation algorithm which particularly excels in balancing energy efficiency and latency. Due to

dynamic scheduling policy this algorithm has become well-suited in the area of IOT/Fog environments(Xu et al., 2021).

In respect to the iFogSim simulation framework, adaptive differential algorithm is integrated to this toolbox so that the task placement across fog/micro-nodes is improved. Task allocations are put into code in such a way that individuals in an initial population are evolved iteratively through adaptive mutation and crossover operations. By frequently analyzing the fitness of the solution based on parameters such as energy consumption the principle drives the population towards adequate structure. Experimental results show that ADE outperforms standard methods such as Differential Evolution(DE), Genetic Algorithm (GA) and Particle Swarm Optimization(PSO) under different workload.
The ADE architecture consists of four parts:

1: **INITIALIZATION AND POPULATION:** The scheduler initiates a population of candidate configurations with each individual representing a possible mapping between the tasks and the available fog nodes. The code usually represents task-to-node mapping. It captures task priority, node identifier, an estimation of the execution time and resource constraints. A diversified population gives wide coverage of the search space.

2: **CROSSOVER AND ADAPTIVE MUTATION STRATEGY**: Unlike the fixed-parameter in case of DE, ADE adapts its mutation factor and and crossover rate after every generation based on the past performance of the preceding populations. This avoids early convergence and keeps the genetic diversity.Mutation adds diversity to individuals and crossover combines traits to further optimize the candidate solutions.

3: **FITNESS TESTING AND SELECTION:** Mapping of each candidate is tested using composite fitness function that takes into account energy consumption, task latency, CPU utilization, and deadlines that should be met by tasks. Those who do not satisfy the constraints are punished and those who are of high quality are reproduced. This will make the solutions converge gradually to optimal scheduling schemes.

4: **CONVERGENCE MONITORING:** ADE has monitoring that is watching the convergence trends and dynamically modifying the mutation/crossover habits in order to retain the exploration capabilities. The process ends when the fitness gains stagnate, or when the limit on the number of generations is exceeded, ensuring convergence without excessive computational cost.

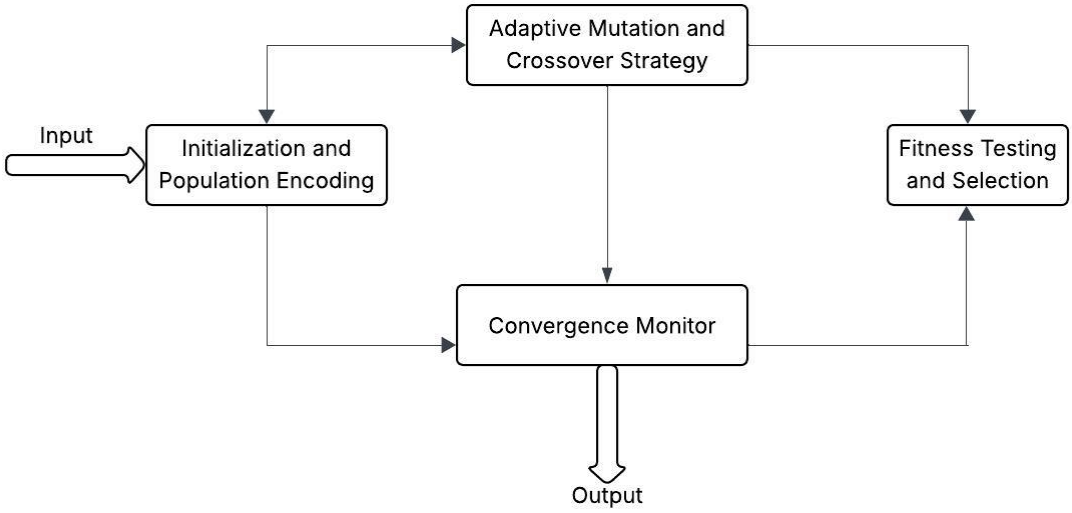

Fig 2: Architecture of Adaptive Differential Evolution algorithm

The flow of Adaptive differential evolution algorithm starts with an input where information such as requirements of the tasks, availability of the resources are fed to the Initialization and Population block where a big pool of candidate solutions is created. Each member has mapping of tasks to the fog nodes.The solution from here is fed into the Adaptive and Crossover Strategy module after initializing it. The mutation and crossover parameters are dynamically adjusted between generations in such a way that the algorithm balances exploration of the solution space and regions that performs well. The newly created offspring are then tested during the block of fitness Testing and Selection where each of them is tested according to energy consumption and task delay. The most successful people are chosen to contribute to next generation. Post that the process enters the Convergence Monitor which examines whether optimization objective is achieved or further evolution is required.The loop is repeated until convergence is achieved.

3.3 Algorithm Flow Diagram

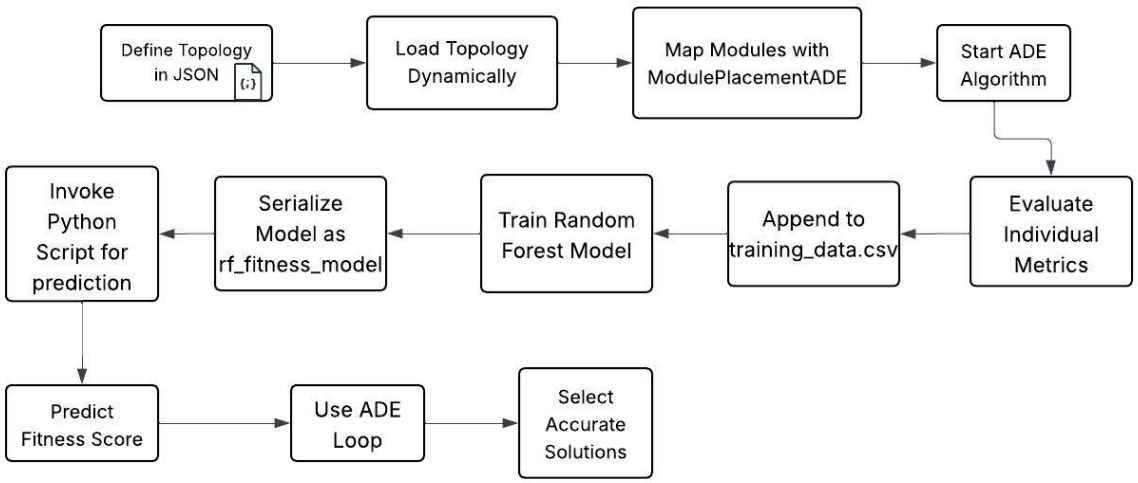

Fig 3: algorithm flow diagram

The flow diagram represents the execution of the Adaptive Differential Evolution (ADE) algorithm in iFogSim simulation environment.It starts with simulation topology in a JSON format that contains the fog layers,especially cloud,proxy,router and sensors. The topology is loaded dynamically into iFogSim and then modules of the application are deployed to different fog devices with the help of custom ModulePlacementADE strategy. After the configuration is done, the ADE algorithm is started.In every execution of the algorithm the performance of the individual solutions will be measured using performance parameters like energy consumption, latency and CPU usage. The values of these metrics are recorded and added to the dataset (training_data.csv) which is later analyzed.

Then the system trains a machine learning model based on the data that was gathered. The training dataset is run through a Random Forest algorithm to determine the non-linear dependencies between input configurations and their fitness score. The python graded model named rf_fitness_model is obtained and is applied in future iterations. When a new set of candidate solutions is produced by the ADE algorithm,a serialized model is obtained which is called by a python script to obtain the fitness values. These scores help the algorithm in choosing the most correct solution. The process is used in the ADE module and it continues to refine the solution set on the basis of the predicted fitness values and the process results in the optimized module placement strategy in the fog environment. This mixed strategy is able to provide smart scheduling decisions at a reduced computation cost in fitness determination.

4 Design Specification

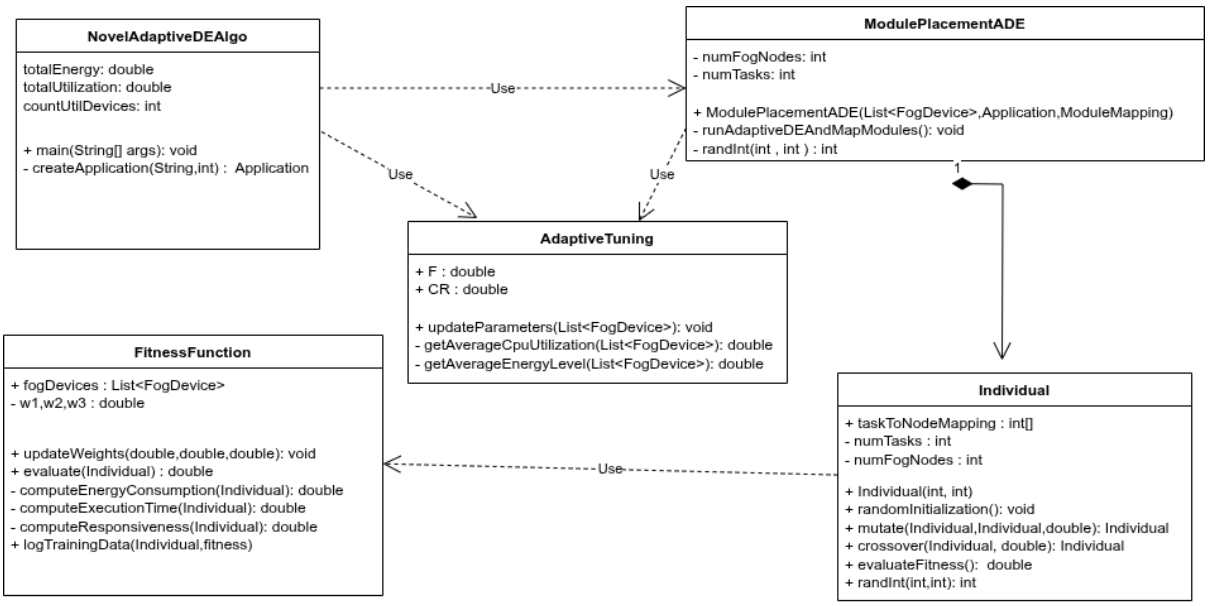

Fig 4: UML diagram representing relationship between the classes.

The Adaptive Differential evolution algorithm has in total 5 java classes.The first class is the **NovelAdaptiveDEAlgo** containing main method which initializes Fogbroker,create application, set up physical topology , helps in mapping modules,starting and ending the simulation. Post this comes the **FitnessFunction** class which evaluates how good a particular individual is in terms of energy consumption, execution time,responsiveness and according to these parameters it assigns a fitness score.The 3rd class is the **AdaptiveTuning** class which dynamically adjusts the adaptive differential evolution algorithm according to the mutation factor and crossover rate.The next class is the **ModulePlacementADE** whose role is to assign

application modules to fog devices in a way that optimizes key performance metrics such as energy consumption and execution time. The last class is the **Individual** class who comes up with a candidate solution when it comes to mapping the application taks to fog nodes. It uses a mutate() function to produce a solution using other individuals mutation factor F and then combines it with the crossover rate.

The above UML diagram represents that there is a dependency of one class on the other.The sketch shows that NovelAdaptiveDEAlgo uses AdaptiveTuning to get the best individuals using the updateParameters() function.Similarly, NovelAdaptiveDEAlgo leverages ModulePlacementADE class which uses the individual who has the best fitness score.It is evident in the diagram that the dependency of one class to other class is shown by dotted line with a tag of **use**.Additionally, NovelAdaptiveDEAlgo is the main orchestrator as it contains the main method. Furthermore, it is evident that the relationship between ModulePlacementADE and the Individual class is of composition as it is clear with a black diamond symbol.It creates,manages objects of Individual class directly with the help of runAdaptiveDEAndMapModules().

5 Implementation

The implementation can be done in two ways. A real-world fog architecture can be constructed by manually configuring the setup or either by using the simulated environment like iFogSim2. The latter method was chosen over the former because it is not feasible to set up the infrastructure due to high cost and maintenance. Additionally, the algorithm is assessed by scaling upto 200 fog devices which is impossible to achieve in live environment. Furthermore, verifying ADE algorithm on real fog nodes can cause overheating problems.

5.1 Algorithm

Algorithm 1: Adaptive Differential Evolution (ADE) for Fog Module Placement

Input:

- $F = \{f_1, f_2, \ldots, f_n\}$: Set of Fog Devices
- $M = \{m_1, m_2, \ldots, m_k\}$: Set of Application Modules
- G_{max}: Maximum number of generations
- Pop_{size}: Size of the population
- F: Mutation factor (initial)
- CR: Crossover rate (initial)

Output: P: Optimal Module-to-Device Mapping

Step 1: Initialization
Initialize population of Pop_{size} individuals with random task-to-node mapping;
Assign fitness to each individual using the fitness function based on energy, time, and responsiveness;
Step 2: Evolution Process
for $g = 1$ *to* G_{max} **do**
  for *each individual* x_i *in population* **do**
    Select three distinct individuals x_{r1}, x_{r2}, x_{r3} such that $r1 \neq r2 \neq r3 \neq i$;
    Mutation:
    $v_i \leftarrow x_{r1} + F \cdot (x_{r2} - x_{r3})$;
    Crossover:
    for *each gene* j **do**
      if $rand(0,1) < CR$ **then**
        $u_{i,j} \leftarrow v_{i,j}$;
      else
        $u_{i,j} \leftarrow x_{i,j}$;
    Selection:
    Evaluate fitness of u_i;
    if $fitness(u_i) < fitness(x_i)$ **then**
      $x_i \leftarrow u_i$;
  Step 3: Adaptive Tuning
  Update F and CR based on current average CPU utilization and energy levels across all devices;
Step 4: Output Result
Select the best individual with the lowest fitness score as the optimal placement;
return P;

Adaptive Differential Evolution (ADE) algorithm is an iterative search algorithm that dynamically assigns application modules to a distributed fog computing system. The algorithm starts with an input which consists of fog devices and application modules. A lot of input parameters are used in the algorithm. These include population size,mutation factor(F), crossover rate (CR) and maximum number of generations (Gmax).During the initialization period a set of individuals are created each of which can represent a potential solution. These individuals are tested based on a fitness function which takes into consideration crucial performance measures like energy consumption,execution time and responsiveness. The algorithm consists of a loop which change and develop over time. From each generation and each individual in the population three different individuals are chosen in order to create a mutant. The original individual is crossed with this mutant using the crossover function to come up with a trial solution. The individual that is trialed replaces the original one if it scores higher in terms of fitness. The algorithm uses adaptive tuning in which mutation factor and crossover rate is changed after every generation. This approach will make the algorithm adaptive to the changing condition of the environment. At the end of the process after running all generations, the individual with the best performance ie the one with the optimized module-to-device mapping is taken as output.

5.2 Technical details

The listed Table 1 depicts tools, which are needed by ifogSim2 as the thesis is developed on top of the framework. All these tools must be integrated successfully to run the simulation.

Technology	Function
eclipse-java-2020-03	Integrated development environment to perform the simulation
Java	ifogsim2 toolkit default programming language
JSON	Used to set the topologies of the IOT/fog setup
ifogSim2	Toolbox which serves as the ground for the simulation

5.3 Modification of framework for Adaptive Differential Evolution

The following are the components of ifogSim2 where changes has been made to execute the adaptive differential evolution on fog nodes.

5.3.1 Customized placement and Test Package

Module placement is an integral part of the proposed algorithm as it will determine how the fog nodes will be placed to boost the performance. Following java classes will be added in the org.fog.placement package.

1. **AdaptiveTuning**: The AdaptiveTuning class changes the value of mutation factor (F) and crossover rate (CR) dynamically in response to the average CPU utilization and energy level of fog devices. This flexibility optimizes the performance of the

Adaptive Differential Evolution (ADE) algorithm as it can adapt to real-time conditions of the system to ensure better use of resources, less energy consumption, and optimal placement of modules in fog computing systems. This factor is crucial to strike a balance between a careful or misuse of the search space.

2. **FitnessFunction**: FitnessFunction class is important because it evaluates hopeful solution in ADE algorithm. It combines energy consumption, execution time and responsiveness together to give a fitness score. These observations are assigned weights such as w1,w2,w3 that can be adjusted to obtain high energy efficiency alongside speed. This combination of solution with weights helps to build a solution which is complete and can be used to train models in future.
3. **Individual**: To assign tasks to fog nodes individual class comes into play. It supports evolutionary actions by using random initialization,mutation,crossover methods to optimize placement strategies. Each case is a simulation of distribution of modules, and over generations, the algorithm advances to better configurations. The sequence of the events during the individual class influences the performance in a dynamic fog environment.
4. **ModulePlacementADE**: ModulePlacementADE class is of primary importance when it comes to implementing ADE algorithm into iFogSim framework. It calls on a number of other Java classes to accomplish this. It communicates with the Individual class to generate, and grow a population of potential task-to-node mapping by mutation and crossover. It also invokes FitnessFunction class to score each mapping on the basis of energy consumption, execution time, responsiveness and gives them some weightage. It also leverage AdaptiveTuning class to dynamically tune the DE parameters F and CR during the run time according to system statistics such as average CPU load and energy consumption. The ADE module placement coordinates these parts in an attempt to achieve optimum placement of modules which improves the energy efficiency, performance and load balancing of fog computing applications.
5. **NovelAdaptiveDEAlgo**: This class is the starting point of the simulation. To begin with,it starts the CloudSim environment,creates fog topology, and starts the application workflow.The createApplication() method is crucial as it is used to create data flows between different fog nodes. The Controller class acts as a co-ordination point that helps in the organization of modules location and the start of the simulation using the ModulePlacementADE strategy. The Actuator class is a mandatory class which performs the final actions on the basis of the processed data which is the final component of the data flow chain. Altogether, they offer adaptive and efficient deployment of fog application.

5.3.2 Change in Utils and Topologies package

A couple of changes were made in org.fog.utils JsonToTopology class especially by making the code more modular and readable by inculcating the switch-case blocks and well named constants. It also added powerful JSON parsing with defensive measures against missing keys and added types to minimize the chance of runtime errors like NullPointerException. Furthermore, the new version includes a lot of logging with descriptive messages so that any

issues can be debugged easily. Additionally it is more efficient in its data structures calling compared to before. The Fog devices such as sensors,actuators are now based on Java stream API's which is now more cleaner and modern. Furthermore, link parsing has been made more stronger than before where validation checks are implemented to ensure that every link has properly formatted source, destination and latency data before connection is tried. Finally, debug output is more clear and descriptive which makes it easier to debug problems in the JSON topology files and it helps to make the overall developer experience much friendlier. All these modifications make the utility more fault-resistant and maintainable in the fog simulation framework.

A lot of changes were done in the routerTopology json file because it plays a vital role in specifying the configuration of our fog computing simulation. It gives a detailed outline of the entire computation nodes, cloud, proxy, router, edge devices for instance cameras and the end devices such as sensors and actuators with parameters such as MIPS, RAM, bandwidth,level of hierarchy and cost of data rate. Additionally, it also specifies how sensors should be distribute. The simulation environment is initialized dynamically using this JSON structure as it more flexible, scalable, since it does not rely on hardcoded infrastructures.

6 Evaluation

With the help of the above mentioned system setup, ifogsim2 simulation is evaluated on different scenarios. The need of executing the simulation multiple times lies in the fact that the algorithm deals with random initialization, mutation and crossover. All these imaginary procedures can yield different outcomes every time during execution. It is so because random initialization creates distinct solutions in the beginning, mutation introduces variations to explore new areas in the solution space and crossover recombines existing solutions in unpredictable ways. All these attributes makes the solution as non-deterministic. Due to this a single run may not accurately represent the algorithm true performance and can depict overestimation or underestimation. To obtain a more statistically reliable and robust result the algorithm is executed roughly 10 times in average and the mean of these observations is typically the output of the solution.

6.1 Performance Metrics

To find effectiveness of the algorithm key metrics such as **energy consumption** and **tuple CPU execution delay** were measured. Energy consumption is an elementary performance metric in IOT/Fog environments which usually represents the total power consumed by fog computing infrastructure during task execution. The assessment of energy consumption gives an understanding of the system efficiency in terms of computational resources. ADE algorithm is employed in such a way that intelligent task placement decisions are made that balance workload distribution & reduce power overhead. The system assigns tasks to appropriate nodes in order to prevent overloading and making network energy efficient.

The second parameter is tuple CPU execution delay. It is nothing but the time which a fog node takes to process an incoming tuple. Even the slightest processing delays in real-time application such as in healthcare monitoring can impact user experience. Therefore it is very crucial to monitor the delay in tuple execution. The ADE algorithm will try to reduce the

CPU delays by determining the best node-task assignments that will make sure that it is executed on time.

6.2 Baseline Techniques

To execute the effectiveness of the algorithm various baseline procedures were used such as DE(Differential Evolution), GA(Genetic algorithm), PSO(Particle Swarm Optimization).Furthermore, these algorithms were compared under varied workloads against the proposed algorithm.

6.3 RESULTS

Algorithm	Energy Consumption	Tuple CPU execution delay
DE	4401076.1733	1.200
GA	4772832.718	1.059
PSO	4408696.185	1.039
ADE	4025692.184	0.650

No of fog devices : 50

Algorithm	Energy Consumption	Tuple CPU execution delay
DE	6254765.976	2.119
GA	6447323.156	2.200
PSO	6042255.284	2.160
ADE	5556768.223	1.109

No of fog devices : 100

Algorithm	Energy Consumption	Tuple CPU execution delay
DE	9974408.177	2.650
GA	9818930.765	2.650
PSO	9635457.771	2.5
ADE	9097852.503	1.660

No of fog devices : 200

The three tables given under the results section take a comprehensive look at the performance of four optimization algorithms with regards to energy consumption and Tuple CPU execution delay. The algorithms namely Differential Evolution (DE), Genetic Algorithm (GA), Particle Swarm Optimisation (PSO) and Adaptive Differential Evolution (ADE) are compared and evaluated under varied workloads starting with 50 fog devices and going upto 200 fog devices. As seen, ADE records the lowest energy consumption in all scenarios and is efficient in the resource-conscious module placement. To illustrate, ADE uses only 4025692.184 units with 50 fog devices compared to the other strategies which are much higher. Furthermore, if the workload is further increased to 100 and 200 fog devices ADE still leads with 5556768.223 and 9097852.503 units of energy consumption respectively, which is very less when compared to other state of art algorithms. This results depict that ADE is far more energy efficient as compared to other algorithms.

The second parameter which is compared is Tuple CPU execution delay. It is a crucial parameter in latency-sensitive IOT applications. The table clearly depicts that the proposed algorithm performs better at small and medium workloads having 0.650 and 1.109 seconds delay at 50 and 100 fog devices respectively. At peak workload it's performance is not degraded, showing 1.660 seconds delay which is a lot less as compared to other algorithms. Although, PSO is little improved as compared to DE and GA in the 200 device setup with 2.5 seconds execution delay it still trails behind ADE. These results depict that our proposed algorithm provides a good trade-off between energy efficiency and computational

responsiveness and proves to be a good option in the fog computing environment where scaling is required.

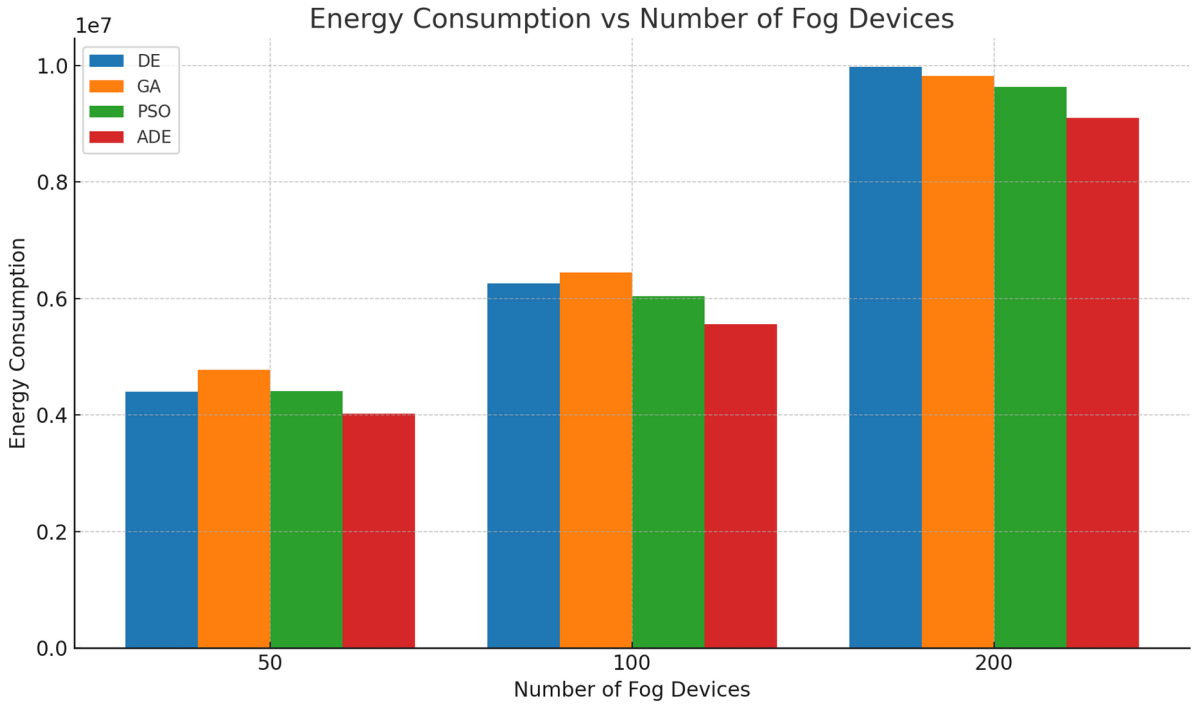

Figure 5: Comparison of Energy Consumption with varying number of fog devices

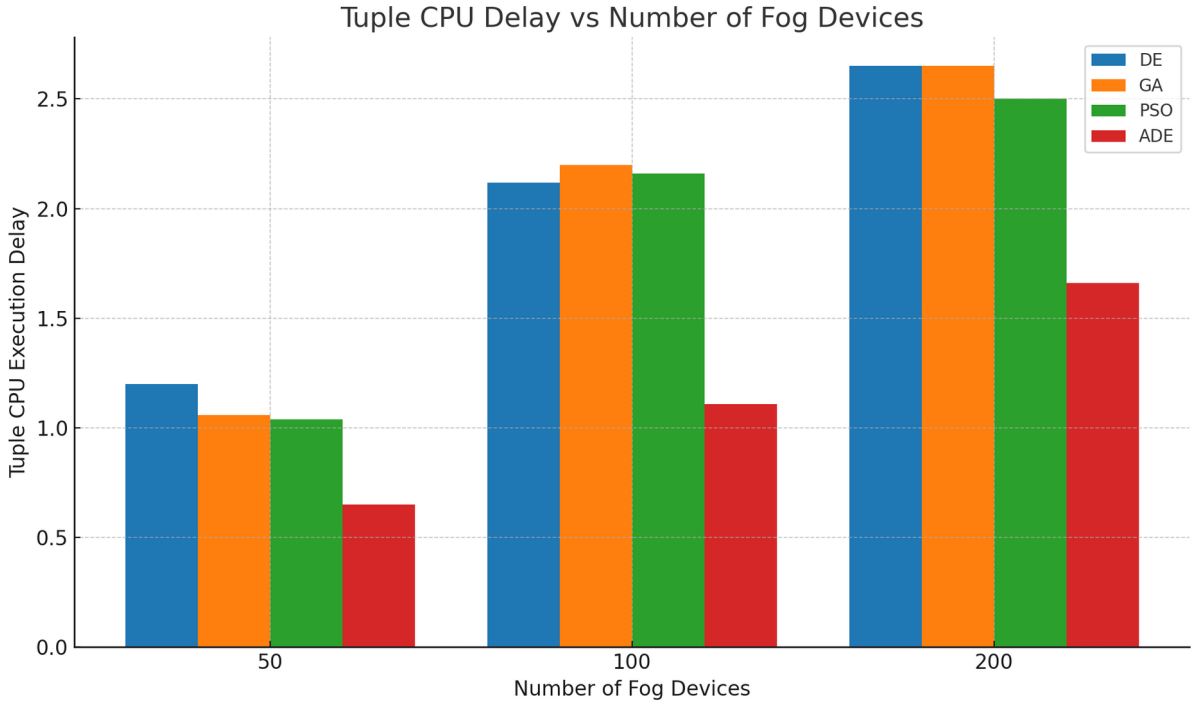

Figure 6: Comparison of Tuple CPU Execution Delay with varying number of fog devices

The above two graphs depicts the energy consumption & Tuple CPU execution delay of four optimization algorithms namely Differential Evolution(DE), Genetic Algorithm(GA), Particle Swarm Optimization(PSO) and Adaptive Differential Evolution(ADE).All these algorithms are evaluated on the topology of 50, 100 and 200 fog devices respectively. From figure 5 it is

clearly observed that as fog devices increases the energy consumption of all algorithms also increases which is due to higher computational and communication load. Among the four, the ADE algorithm consumes the least energy across all three configurations. When there are 50 fog devices the energy consumed by ADE is approximately 4 million Jules whereas GA consumes the highest with 4.77 million. The same pattern repeats at 100 and 200 fog devices also. This portrays the effectiveness of ADE in managing resources efficiently in a fog computing environment especially during the time when workload increases.

Figure 6 demonstrates the relationship between Tuple CPU Delay & no of fog nodes wherein DE,GA,PSO and ADE algorithms are compared on same workload described in the former sketch. The graph depicts an upward trend in CPU delay as the number of devices increases, which is evident from the fact that complexity increases when workload increases. The ADE algorithm is better than the others as it demonstrates the lowest delays across all configurations. It registers a delay of 0.65 seconds at 50, 1.109 at 100 and 1.66 at 200. In contrast, DE and GA have higher delays with a maximum of 2.65 at 200 devices. PSO also follows a similar trend but performs a little better than DE and GA. This makes ADE compatible for latency-sensitive fog computing applications.

6.4 Discussion

The above statistics depicts that the proposed algorithm outperformed other state of the art algorithms in both Energy Consumption and Tuple CPU Delay. When the simulation is performed at maximum workload ie with 200 fog devices ADE attained the lowest energy consumption which was approximately 9.09 million Joules whereas other algorithms such as DE,GA achieved roughly 9.97 million Joules. Statistically, ADE surpassed others by 8.83 %. Not only did ADE performed well in Energy Consumption but also in TupleCPUDelay. It attains the lowest tuple CPU delay of 1.66 seconds,however DE and GA attained a delay of 2.65. It was able to achieve such lower energy consumption due to its self-adaptive,mutation and crossover strategies .These strategies helped it to converge faster to best solutions as compared to others. Other algorithms showcased higher energy usage due to their slower convergence due to fixed parameters.

On the contrary, due to randomness the suggested algorithm was required to be executed many times continuously and the mean of 10 observations was expected to be the result of the simulation. In future if any method could reduce randomness in algorithm, it will then produce even better results. Furthermore, in the simulation the algorithm was evaluated in a fixed topology which consisted of 1 cloud node,1 proxy,1 router and fog devices. The result would greatly change if the configuration of the topology is updated. For instance, instead of only 1 router which collects and send messages to different fog devices , more routers were installed to make the flow seamless and better. Furthermore,if more than 1 proxy servers were introduced in the structure than results would greatly change. Additionally, real time physical device offloading and security implementation is not incorporated so network reliability and data-privacy domains are left unexplored.

7 Conclusion and Future Work

The main question that this research aimed to address is how to use Adaptive Differential Evolution (ADE) algorithm to optimize resource allocation and task scheduling in a fog computing environment particularly with the intent of reducing energy consumption and increasing execution efficiency when compared to conventional methods such as Genetic Algorithm (GA) and Differential Evolution. The simulation outcomes revealed that the suggested ADE-based scheduling framework is more energy and processing delay efficient as compared to DE and GA. In particular, it was found that ADE consumed around 9.09 million joules which is roughly 8.83% less energy than DE and GA which consumed around 9.97 million joules. Moreover, the TupleCPU Execution Delay of ADE displayed a consistent level of performance when the workload was increased. This showed that the algorithm is very flexible.

However, despite these upsides ADE has many limitations. Firstly, the entire simulation topology is homogeneous consisting of 1 cloud,1router,1proxy and varying fog devices according to workload. There is no consideration of heterogeneous topology structure such as numerous router with various proxies connected to various fog devices. Furthermore, no real-live testing is performed .It is so because testing IOT and fog devices can be very expensive and complex because different components of the infrastructure can incur hugh cost because the setup required for the simulation is based on scalability. For instance in our setup we are testing simulation on 50,100 and 200 fog devices. In real-life scaling is not possible to achieve due to limitation in appliances. Additionally, a lot of overheating can occur when scaling devices which can eventually lead to data loss.

The limitations can be solved by implementing heterogeneous fog topology with practical offloading which would inculcate fault tolerance and security. In addition to that, ADE algorithm can be further improved by integrating QoS-aware or security-aware scheduling.

References

Hussain, F., Hussain, R., Hassan, S.A. & Hossain, E. (2021) *Adaptive DE Algorithm for Novel Energy Control Framework Based on Edge Computing in IIoT Applications*. IEEE Transactions on Industrial Informatics. **CORE A**.

Abbas, A., Zhang, Y., Taherkordi, A. & Skeie, T. (2020) *Towards an automatic deployment model of IoT services in Fog computing using an adaptive differential evolution algorithm*. Journal of Systems Architecture, 110, pp.101799. **CORE B**.

Benhida, K., Kettani, D. & El-Khatib, K. (2021) *A predictive energy-aware scheduling strategy for scientific workflows in fog computing*. Future Generation Computer Systems, 117, pp.379–393. CORE A.

Jiménez, T., et al. (2019) *Fog-driven context-aware architecture for node discovery and energy saving strategy for Internet of Things environments*. Journal of Parallel and Distributed Computing, 134, pp.30–41. **CORE A**.

Yousefpour, A., et al. (2018) *On reducing IoT service delay via fog offloading*. IEEE Internet of Things Journal, 5(2), pp.998–1010. **CORE A**.

Rego, P., Rodrigues, J.J., Lloret, J. & de la Torre, I. (2022) *A novel fog-driven context-aware architecture for energy efficiency in IoT environments.* Journal of Network and Computer Applications, 205, pp.103392. **CORE A**.

Sharma, A., et al. (2021) *Efficient job scheduling paradigm based on hybrid sparrow search algorithm and differential evolution optimization for heterogeneous cloud computing* plaforms.Journal of Systems and Software, 182, pp.111063. **CORE A**.

Kebande*, V.R., et al.* *(2022) A novel distributed fog-based networked architecture to preserve energy in fog data centers. Sensors, 22(15), pp.5793. Not Ranked by CORE, ~20 citations on Google Scholar.tforms*. Journal of Systems and Software, 182, pp.111063. **CORE A**.

Kang, J., et al. (2021) *A Novel Fog-Based Multi-Level Energy-Efficient Framework for IoT-Enabled Smart Environments*. IEEE Access, 9, pp.20367–20378. **CORE B**.

Djumanazarov, K., et al. (2020) *Adaptive Energy-Aware Computation Offloading for Cloud of Things Systems*. Future Internet, 12(10), pp.163. Not Ranked by CORE, ~30 citations on Google Scholar.

Botta, A., et al. (2016) *Fog Computing May Help to Save Energy in Cloud Computing*. IEEE Cloud Computing, 3(2), pp.32–38.**CORE B.**