Dynamic Resource Management of Fog-Cloud Computing for IoT Support

Mariia Surmenok
San Jose State University

Dynamic Resource Management of Fog-Cloud Computing for IoT Support

A Project Report

Presented to

The Faculty of the Department of

Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By

Mariia Surmenok

December, 2021

The Designated Project Committee Approves The Project Titled

Dynamic Resource Management of Fog-Cloud Computing

for IoT Support

By

Mariia Surmenok

Approved For The Department Of Computer Science

San José State University

Dec 2021

Dr. Teng Moh Department of Computer Science

Dr. Melody Moh Department of Computer Science

Dr. Kong Li Department of Computer Engineering

ABSTRACT

The internet of things (IoT) is an integrated part of contemporary life. It includes wearable devices, such as smart watches and cell phones, as well as sensors for Smart City. Fog computing can improve the efficiency and battery life of IoT devices by offloading tasks to fog cloud. It is important to have fog clusters near the IoT device for faster data offload. The goal of this project is to develop dynamic resource allocation for on-demand fog computing cluster to efficiently deploy tasks from IoT. This report studies the different research papers about the current state of resource management in cloud environment. It overviews the main mechanisms, objectives, and the evaluation criteria of the state-of-the art solutions. This report discusses the results of different modifications of memetic algorithm. In our project, we try to minimize the task completion delay, number of requests failed by deadline for all services and services with the high priority by finding the closest to a user fog node that has enough available resource. In this project we will use Yet Another Fog Simulator (YAFS) for simulating and testing the effectiveness of proposed memetic algorithms modifications.

Keywords – IoT, Fog Computing, Resource Management, Mobile Cloud Computing

TABLE OF CONTENTS

I. Introduction 1

II. Literature Review 3

A. Resource Management in Cloud Computing Environment 3

B. Resource Management in Fog and Peer-to-Peer environment 6

III. Architecture 11

IV. Problem Formulation 14

V. Algorithms 18

A. Baseline algorithms 18

B. Memetic Algorithm Modifications 21

C. Machine Learning Optimization 27

VI. Simulation 29

VII. Results and Analysis 35

A. Metrics. 35

B. Results and Discussion for Configuration with Average Network and Applications Settings 36

C. Results and Discussion for Configuration with Different Requests Frequency 51

D. Results and Discussion for Configuration with Different Fog Devices' Resources 55

E. Summary of the Best Memetic Modifications 58

VIII. Conclusion and Future Work 60

References 63

LIST OF TABLES

Table 1. Summary of Resource Management Methods in Cloud and Fog Computing. 4
Table 2. Comparison table of different combinations of modifications created for memetic baseline algorithm. 22
Table 3. Experiment scenarios. 30
Table 4. Application settings.. 31
Table 5. Fog devices and cloud settings.. 32
Table 6. Application settings for configuration with requests frequency above average. 32
Table 7. Application settings for configuration with requests frequency below average. 33
Table 8. Fog devices settings for configuration with fog devices' resources above average. 33
Table 9. Fog devices settings for configuration with fog devices' resources below average. 33
Table 10. Results of 100 experiments for scenario with 20 fog devices and 50 services. 36
Table 11. Results for 100 experiments for scenario with 40 fog devices and 100 services. 37
Table 12. Comparison of results for Experimental 2 with baseline algorithms of 100 experiments for scenario with 20 fog devices and 50 services. Average configuration. 41
Table 13. Comparison of results for Experimental 2 with baseline algorithms of 100 experiments for scenario with 40 fog devices and 100 services. Average configuration. 42
Table 14. Comparison of results for Experimental 6 with baseline algorithms of 100 experiments for scenario with 20 fog devices and 50 services. Average configuration. 43
Table 15. Comparison of results for Experimental 6 with baseline algorithms of 100 experiments for scenario with 40 fog devices and 100 services. Average configuration. 44
Table 16. Comparison of results for Experimental 3 with baseline algorithms of 100 experiments for scenario with 20 fog devices and 50 services. Average configuration. 44
Table 17. Comparison of results for Experimental 3 with baseline algorithms of 100 experiments for scenario with 40 fog devices and 100 services. Average configuration. 44
Table 18. Comparison of results for Experimental 4 and Experimental 5 with baseline algorithms of 100 experiments for scenario with 20 fog devices and 50 services. Average configuration. 45
Table 19. Comparison of results for Experimental 4 and Experimental 5 with baseline algorithms of 100 experiments for scenario with 40 fog devices and 100 services. Average configuration. ... 45
Table 20. Comparison of results for Experimental 7 and Experimental 8 with baseline algorithms of 100 experiments for scenario with 20 fog devices and 50 services. Average configuration. 46
Table 21. Comparison of results for Experimental 4 and Experimental 5 with baseline algorithms of 100 experiments for scenario with 40 fog devices and 100 services. Average configuration. ... 47

Table 22. Comparison of results for Experimental 9 with Experimental 8 and baseline algorithms of 100 experiments for scenario with 20 fog devices and 50 services. Average configuration. 48
Table 23. Comparison of results for Experimental 9 with Experimental 8 and baseline algorithms of 100 experiments for scenario with 40 fog devices and 100 services. Average configuration. ... 48
Table 24. Simulation results for traffic above average for scenario with 20 fog devices and 50 services. 52
Table 25. Simulation results for traffic above average for scenario with 40 fog devices and 100 services. 52
Table 26. Simulation results for traffic below average for scenario with 20 fog devices and 50 services. 53
Table 27. Simulation results for traffic below average for scenario with 40 fog devices and 100 services. 54
Table 28. Simulation results for settings with fog devices above average for scenario with 20 fog devices and 50 services. 55
Table 29. Simulation results for settings with fog devices above average for scenario with 40 fog devices and 100 services. 55
Table 30. Simulation results for settings with fog devices below average for scenario with 20 fog devices and 50 services. 56
Table 31. Simulation results for settings with fog devices below average for scenario with 40 fog devices and 100 services. 56
Table 32. Comparison of best performing modifications with Memetic baseline. 58

LIST OF FIGURES

Figure 1. Proposed three-layer architecture with volunteering fog devices. 13
Figure 2. Flow chart for Baseline Memetic algorithm. 21
Figure 3. Comparison of memetic algorithm modifications. Rectangles are placement algorithms. An arrow shows what the base algorithm for each modification is. An ellipse shows the changes introduced in the algorithms enclosed in the dashed box. 26
Figure 4. Average calculation time for modifications of memetic algorithm. 38
Figure 5. Average total response time. 38
Figure 6. Average percent of failed requests. 39
Figure 7. Average total response time for services with high priority. 39
Figure 8. Average percent of failed requests for services with high priority 40
Figure 9. Average number of active devices for scenario with 20 fog devices and 50 services. ... 40
Figure 10. Average number of active devices for scenario with 40 fog devices and 100 services. 41
Figure 11. Comparison of average calculation time for memetic baseline, memetic without local search, and two memetic algorithms that perform local search every second generation. 50
Figure 12. Average total response time for settings with traffic above average. 52
Figure 13. Average percent of failed requests for settings with traffic above average. 53
Figure 14. Average total response time for settings with traffic below average. 54
Figure 15. Average percent of failed requests for settings with traffic below average. 54
Figure 16. Average total response time for setting where fog devices' resources above average.. 56
Figure 17. Average number of active fog devices for scenario with 20 fog devices and 50 services where fog devices' resources are below average. 57
Figure 18. Average number of active fog devices for scenario with 20 fog devices and 50 services where fog devices' resources below average. 57

I. INTRODUCTION

IoT devices are ubiquitous in contemporary life. According to [1], there will be 35 billion devices by the end of 2021. IoT devices include wearable devices, mobile phones, Smart City sensors, and others. These devices generate a lot of data which usually requires a lot of processing capacity. However, sending the data to the remote data center or remote fog cloud can take a lot of time, decreasing the quality of service and is not suitable for time-sensitive applications. Moreover, sending the data to the remote host can drain the battery of the IoT device. Having a fog cloud near the device can alleviate these challenges while providing sufficient computational power for tasks generated by IoT devices.

Far away fog cluster may not be very beneficial for the efficient support of IoT devices. Instead, we need a fog cluster near the IoT device. It can be achieved by creating a fog cluster dynamically when there's a need for it in a particular location. For example, Sami and Mourad [2] proposed on-demand formation using volunteering devices as fog nodes. In such a setting, resource allocation is the main challenge. The main goal of resource management in on-demand fog formation is to identify the best host in terms of location, resource availability, and capability to complete task in given deadline.

In this project we will use memetic algorithm proposed by Sami [2], and introduce our own modifications. Memetic algorithm (MA) is a modified version of genetic algorithm (GA) which is inspired by nature and simulate natural selection, and unlike GA, memetic al is not prone to premature convergence because of local search.

This report is organized in the following way. Section 2 summarizes the related work of resource management in cloud and fog computing. In section 3 the architecture and its main parts are discussed. Section 4 describes problem formulation and objective functions. Section 5 explains baseline algorithms and the modification of memetic algorithm that are used for this project. In section 6 we explain simulation setup and two simulation scenarios. In section 7 we discuss experiment results. Finally, paper concludes with summary and related work in section 8.

II. LITERATURE REVIEW

Several architectures can help to support IoT with additional computational power, such as traditional cloud computing, standalone fog computing, fog computing with additional support from the cloud, and architectures that use peer-to-peer technology. The summary of all the approaches is presented in Table 1.

A. Resource Management in Cloud Computing Environment

Cloud computing represents traditional datacenters. It remains a popular choice to support IoT devices. The introduction of containerized virtualization and the interest in energy and cost optimization inspire recent research.

Zhong and Buyya [3] created a modified Kubernetes orchestrator to optimize the heterogeneous cloud computing data center's cost. For dynamic resource management, they employed live task migration via CRIU and task-packing using best fit decreasing. The authors considered two types of jobs, long-running and batch jobs, and scheduled them based on task kind and its completion time. The decision for migration is policy-based and triggered when utilization of the host false below 50% threshold. The authors tested their approach on the Australian National Cloud Infrastructure (Nectar). They used two types of synthetic applications and four kinds of workload: stable, growing, cycle, on-and-off. They compared their approach with Stratus and COCA and proved that their approach reduces the overall cost by 23-32%. Their approach tries to reach the best QoS but does not prevent violation of QoS requirements for applications.

Table 1. Summary of Resource Management Methods in Cloud and Fog Computing.
ML machine learning, *BF* best fit, *ARIMA* autoregressive integrated moving average, *TSMM* two-sided matching method, *MCDM* multi criteria decision matrix, ILP integer linear programming, DDPG Deep Deterministic Policy Gradient, *GA* genetic algorithm, *ACO* ant colony optimization, *MA* memetic algorithm, *DNN* deep neural network, *FF* first fit, *DQL* Deep-Q-Learning, *A3C-R2N2* Asynchronous-Advantage-Actor-Critic Residual Recurrent Neural Network.

Publication	Methods Used			Objectives			
	ML	Metaheuristic	Heuristic	Energy	Time	Utilization	Dynamic Changes
Cloud Type: Centralized Cloud							
Zhong [3]			BF			✓	
Zhang [4]			ARIMA and TSMM	✓	✓	✓	
HeporCloud [5]			statistical	✓	✓		
Gholipour [6]			MCDM	✓		✓	
Cloud Type: Fog							
KEIDS [7]			ILP	✓		✓	
Naha [8]			ranking		✓		✓
Ren [9]		GA and ACO		✓		✓	✓
Sami [2]		MA			✓	✓	
Wu [10]	DNN			✓	✓		
Chen [11]	DDPG			✓	✓		
RLSK [12]	DQL					✓	
Tuli [13]	A3C-R2N2			✓	✓		

Zhang et al. [4] proposed an energy-aware framework for two-tier virtualized heterogeneous cloud data centers. In their work, authors consider only container migration. The restriction for container placement is that only the containers belonging to the same job can be hosted on the same virtual machine. This rule should provide an additional security level in case some of the containers from other jobs are compromised. Authors used an energy model for problem formulation, including the overhead of creating a virtual machine and the SLA metric. The framework had to solve several tasks for the initial placement and dynamic consolidation at runtime, detecting underloaded and overloaded VMs and hosts and making a decision for container migration. To find the best hosts for VMs, the authors used the many-to-one two-sided matching method where both VMs and hosts calculate the coefficient to find the subset of the desired pair. For underloaded and overloaded host and VMs detection, the authors employed the ARIMA algorithm to predict the resource usage in the nearest future to prevent unnecessary container migration. The authors evaluated their framework using ContainerCloudSim simulation and workload traces from PlanetLab. Their approach outperforms the combination of classical algorithms and state-of-the-art methods in terms of energy consumption by at least 13.8%. However, the overall complexity of the authors' approach is O(n2) and may not be suitable for time-sensitive real-time jobs.

Gholipour et al. [5] also used two-tier virtualization. Their joint VM and container consolidation algorithm is optimized for energy-aware resource management. The algorithm aims to place containers in the minimum number of virtual machines and the smallest number of physical servers. Unlike other researchers, the authors considered the joint VM and container consolidation policy to identify whether a virtual machine or a container should be migrated. Resource correlation is calculated to find which virtual machine causes the overloading of the server. To choose whether a virtual machine should be migrated, the multi-criteria decision matrix is used. If the candidate VM is not selected for migration, the containers placed on this virtual machine are migrated. To evaluate their approach, the authors run a simulation on ContainerCloudSim with workload traces from PlanetLab. The experimental results showed that their approach reduces the energy consumption, SLA violation, and a number of migrations comparing to the state-of-the-art algorithms.

While previous authors [3] used containerization or two-tier technology where containers are placed inside virtual machines [4][5], Khan et al. [6] proposed the combination of different virtualization technologies inside the same framework. There are four types of platforms in their framework simultaneously, such as bare-metal, containers over virtual machines, and others. The authors explain that some jobs are running faster using particular virtualization technology. The authors used the ERP metric (energy response time per product) for the problem formulation, which expresses energy consumption and SLA metric. Orchestrator places jobs on a particular platform based on statistical methods finding similar jobs from the past. The migration decision for containers or virtual machines is policy-based and triggered when a specific threshold is reached. However, the migration is not initiated if the predicted remaining runtime is too small. The authors used CloudSim simulator and workload traces from Intel, Microsoft Azure, and Google to evaluate their approach. They found out that their method can reduce energy consumption by 14-37% compared to single virtualization technology.

The papers discussed above use a policy-based algorithm for initial placement and statistical approaches for predicting future workload and similar jobs. Most of these papers prefer containers over virtual machines because they are much lighter and easier to deploy. All the authors consider container migration during runtime, although creating an overhead can significantly improve resource utilization.

B. Resource Management in Fog and Peer-to-Peer environment

Fog computing was first introduced by CISCO in 2012. The main idea is to have some processing power, fog nodes, closer to edge devices.

Kauk et al. [7] proposed a Kubernetes-based scheduler to provide an edge-cloud ecosystem for industrial IoT devices. The goal is to reduce carbon footprint, optimize energy consumption, and to improve performance by minimizing the inference among co-located containers. To reduce the interference, the authors proposed to place similar containers into the same hosts identifying jobs either as CPU or network intensive. The authors wanted to maximize the use of available green energy resources and minimize the number of active hosts. The authors expressed the problem as an integer linear programming problem and used Mosek solver to find an optimal solution. For the experiments, they simulated four different clusters and used workload traces from the Google dataset. The solution was tested against FCFS and some state-of-the-art algorithms and showed better results in maximal use of green energy, minimization of interference, and overall energy minimization. For constraints, the authors used CPU, memory, and network availability but didn't consider QoS. Also, it is unclear if they included the network overhead or host location into their energy model.

Naha et al. [8] proposed an algorithm for resource allocation in a fog environment with dynamic user requirements, such as changing deadlines. Their solution includes ranking available resources in fog and cloud and rule-based provisioning. For their experiments, authors used CloudSim and synthetic workload. Their solution outperforms the performance of resource-aware and latency-aware algorithms for a given setup. Still, it is unclear if it is more efficient against other approaches, such as meta-heuristic genetic algorithms.

Some recent research employs a genetic algorithm approach to solve the resource allocation problem. Ren et al. [9] proposed a hybrid algorithm to reduce energy consumption in the fog environment for IoT devices. They suggested the combination of Genetic (GA) and Ant Colony Optimization (ACO) algorithms. These algorithms run in parallel and update each other values after each iteration. Makespan, energy consumption, and cost are used for the fitness function. Also, the dynamic requirements, where the user requests the decreasing deadline, are considered. For the evaluation, the authors used the CloudSim toolkit with a synthetic dataset. They run multiple scenarios where the number of VMs varies between 20 and 2000 and the number of physical costs between 8 and 800. The authors verified the stability and convergence of the proposed method. The hybrid algorithm outperforms GA and GA-CSO algorithms in terms of energy, time, and cost.

Sami and Mourad [2] also employed the genetic algorithm for their framework. The proposed on-demand fog formation using volunteering devices near the IoT device or user. The cloud maintains a database with the recent information about volunteers, including available resources, time availability, and location. The orchestrator is created on one of the volunteering devices or, if the job is time-sensitive, the orchestrator is placed in the cloud for faster fog deployment. For the proposed approach, the authors used kubeadm and Docker containers so that any volunteer can download the required images from Docker Hub. The container placement problem is solved using a memetic algorithm. The goal is to place services on devices with the best time availability and proximity to the user while maintaining the best QoS. The authors compared their container placement strategy with the first-fit on a long time and the first-fit on short distance algorithms. The proposed approach shows a better response time with the increasing number of requests. The authors evaluated the architecture, using AWS instance as a cloud and computers in the lab for fog formation. They compared the response time for an increasing number of requests and proved that dynamic fog formation outperforms the cloud, static fog, and remote fog. Although the proposed approach showed the best response time and scalability, security is the primary concern in this paper.

Wu et al. [10] used Deep Neural Network (DNN) for making the decision for task offloading for the heterogeneous cloud. The goal is to minimize the task completion delay and energy consumption. The framework consists of multiple DNN that share the same database, and the best result updates the database. The experiments show that their approach outperforms the traditional offloading schemes. However, they do not consider that the time for offloading tasks into central and fog clouds differs in their experiments. Also, they assume that the cloud and fog cloud has unlimited computational power, which is not always valid for fog.

[11], [12], and [13] use deep reinforcement learning for optimizing tasks placement. Chen et al. [11] used Deep reinforcement learning for dynamic resource management of joint power control and resources for Mobile Edge Computing (MEC). Assuming that IoT has limited battery capacity, it is essential to considered transmission delay over a wireless network and battery capacity. It is important to minimize the long-term processing delay. The authors formulated the problem as a Markov decision problem and used Deep Deterministic Policy Gradient (DDPG) to find the optimal offloading scheme. To evaluate their policy, the authors simulated one MEC with 25 IIoT devices and a synthetic dataset. They compared their scheme with A3C, URM, and RRM and proved their approach can improve the average transmission delay by 4-17%.

Huang et al. [12] used Deep Reinforcement Learning to schedule jobs in federated Kubernetes clusters. Their Deep Q-Learning model was trained to schedule batch jobs between multiple homogeneous clusters. The goal is to balance the average utilization among clusters and the average utilization of each resource within each cluster, preventing bottlenecks. For the experiment, they used three homogeneous clusters and simulated workload. The authors compared the result with traditional scheduling algorithms, such as First Fit, Round-Robin, and Least Load. The proposed approach was evaluated with the following metrics: resource utilization within a cluster, the utilization between different clusters, and maximum completion time in each cluster. The result significantly outperforms the traditional algorithms in load balancing and utilization with a slightly greater makespan. Their approach showed better adaptability to changing workloads. However, they used very specific environments, homogeneous hosts, and batch jobs only and compared them with very basic algorithms. Also, in their approach, if the job cannot be scheduled at the current moment, they postpone it to some random time in the future, which will not guarantee that it will be scheduled eventually.

Tuli et al. [13] also used Deep Reinforcement learning for scheduling tasks in a fog environment. Tuli et al. included the mobility factor into their research and accounted for changing resource and bandwidth requirements for the service. They utilize Policy gradient-based Reinforcement learning method (A3C) to accelerate the learning. In their architecture, the authors used multiple actor-agents at the same time. Each agent has its own neural network and is responsible for its own set of fog nodes allowing it to train networks in parallel. The agents update the shared global parameters, which accelerates the exploration of larger state-action space—using a residual recurrent neural network allowed to approximate function from state to action and find patterns in the data to predict the future workloads. The authors used energy consumption, average response time, cost, and SLA violation to evaluate the efficiency of their approach. They performed experiments using iFogSim and CloudSim using an open-source Bitbrain dataset. Their results show that their approach was 14.4%, response time by 7.74%, SLA violations by 31.9%, and cost by 4.64%. However, their architecture is designed for a fixed number of edge nodes and tasks and needs future work to enable scalability.

Resource management in fog computing is concerned with effective offloading and task scheduling for a dynamic heterogeneous environment. Two main approaches are to use evolutionary algorithms, such as genetic algorithm and memetic algorithm, and deep reinforcement learning approach. The introduction of A3C allows to train scheduler faster and enable to find temporal patterns in workload.

III. Architecture

The dynamic on-demand fog formation using volunteering devices is an interesting approach to help create fog near IoT devices. It is a peer-to-peer approach with a database in the central cloud that keeps track of all peers involved in fog devices in a given location. The architecture of the proposed environment consists of three layers. The first layer is users and various IoT devices that produce request and send it via edge devices such as wi-fi routers. The second layer has multiple fog devices. Fog devices are computers near the users that rent their resources. The third layer is the centralized cloud with virtually unlimited resources but is placed far away from users. The proposed architecture is depicted in Figure 1. The main parts of the architecture are described below.

Fog devices. Fog devices are simple machines that individuals or third-party businesses are willing to lend to create a fog cluster in a specific location. Participants who want to lend their machines as fog devices apply to be volunteers and send the resource information about their machines, including resource information, device location, and time availability, to the cloud. Volunteering fog devices periodically send an update about their current state. All this information is stored in a database on the cloud and used when a fog cluster needs to be created in a particular location. We have information about instructions per time interval (IPS), memory (RAM), disk storage, number of cores available, its x and y coordinates, and time availability for each fog device.

Users. Users are the entities that create tasks that need to be processed. In this project, we assume that our users are some IoT devices, for example, sensors and smartwatches, that create computationally heavy tasks. These IoT devices want to process tasks remotely to save battery and reduce computational time. Users are connected to gateway devices through which users send requests with tasks. Multiple users can be connected to the same gateway device. These gateway devices do not process requests themselves but send requests to fog devices or directly to the centralized cloud.

Application. Application is software that has two services, client-side and server-side. The Client-side is placed on an IoT device (user). Server-side service is placed either on fog-device or in a centralized cloud. Each application has an id, resource requirements, priority, and deadline. There may be multiple applications sent through the same gateway. The Client-side periodically sends a request with tasks to the server-side. A request has information about the number of instructions and number of bytes. Later in this report, we will refer to server-side service as a service. For each service to be placed on fog, we have information about resource requirements for memory (RAM), disk storage, number of cores, deadline, and this service priority.

Orchestrator. Initially, there is no fog cluster in a particular location. When the need for a fog cluster in a given location arise, an orchestrator is created on one of the volunteering fog devices in this location. The replica of the orchestrator is placed on another volunteer device to improve fault tolerance. If the fog device hosting the local orchestrator runs out of available time, another fog device is chosen, and local orchestrator migration is performed. Cloud shares information with the local orchestrator about all available volunteers at this location that can be used as fog devices. This information includes volunteering fog devices' physical location and resource availability. Using information received from the cloud database, the local orchestrator performs the placement of server-side services for all applications needed in this location. Orchestrator does not know network topology. However, it has information about each fog location and can estimate the physical distance from the user to each fog device. This project will use various placement algorithms to perform the initial placement of all the services in all available fog devices. If the placement algorithm cannot find a suitable fog device in the fog cluster, it sends this service to the centralized cloud, so all requests for this application are processed on the cloud.

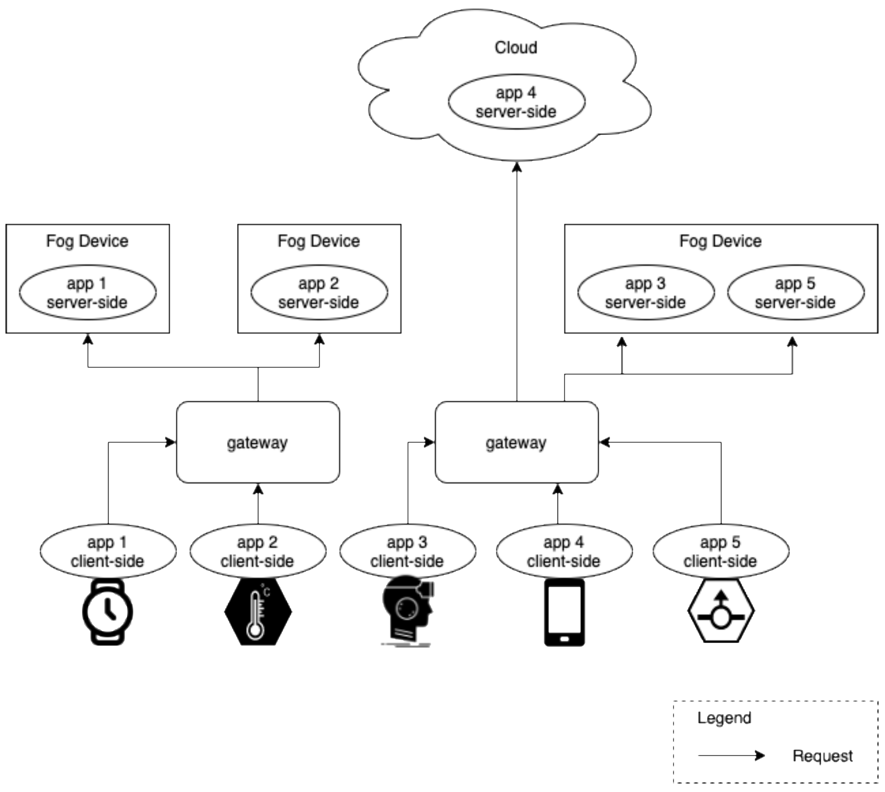

Figure 1. Proposed three-layer architecture with volunteering fog devices.

IV. Problem Formulation

We use problem definition similar to [2] with our own addition of deadline, x and y coordinates, and our own representation of placement as vector instead of matrix proposed by Sami.

Problem definition. We have a set of services $S = \{s_1, s_2, \ldots, s_n)$ needed to be placed close to usersand fog devices $D = \{d_1, d_2, \ldots, d_m\}$ who can host services in a given location. We have number of services *n*, and number of fog devices *m*. Services to be placed are represented as a matrix where each row is one of the service characteristics. Each service has the following characteristics:

$$S_i = [S_{cpu}, S_{mem}, S_{disk}, S_{priority}, S_{deadline}, S_x, S_y]$$

where:

S_{cpu} : number of CPU required

S_{mem} : number of memory required

S_{disk} : amount of storage space required

$S_{deadline}$: deadline for tasks associated with this service

S_x : x coordinate of a user for this service

S_y : y coordinate of a user for this service

Each fog devices has the following characteristics:

$$Dj = [D_{cpu}, D_{mem}, D_{disk}, D_{time}, D_x, D_y, D_{IPT}]$$

where

D_{cpu} : number of CPU available

D_{mem} : number of memory available

D_{disk} : number of storage available

D_{time} : time when fog device available to host services

D_x : x coordinate of fog device

D_y : y coordinate of fog device

D_{IPT} : speed of processor

For speed of processor, D_{IPT}, we assume that for the same fog device all processors have the same speed.

We want to achieve the best placement of services on the available volunteering fog devices to satisfy multiple objectives and without violating constraints such as resource availability on fog devices. The solution is represented as a vector Ki with length n where $K_i \in (0, m]$ indicating the id of fog device for each service.

Constraints. For our placement algorithms, the following constraints should not be violated. First, the resource requirements of all services placed on a fog device, should not exceed the resource capacity of this fog device.

Service should be placed maximum on one device. It is naturally preserved by representing solution as a vector instead of matrix used in [2].

Objective functions. We use the five objective functions introduced in [2] with modification for objective 4, host distance minimization. We also introduce two more objective functions and test their performance in separate modifications of the memetic algorithm.

Objective 1, *F1*: maximize the number of services on fog devices. We count all the services placed on the fog cluster in the current location. If service was not placed on any fog devices in the fog cluster, we assume that service was sent to the cloud.

Objective 2, *F2*: maximize the number of services with maximum priority on fog devices. We count all the services with the high priority placed on fog.

Objective 3, *F3*: maximize the total time availability by choosing fog devices with the biggest time available. For all services placed on fog, we sum the time availability of fog device that host each service.

Objective 4, *F4*: minimize distance between user and service by choosing the closest fog device. The original paper used a single distance value to characterize the distance between fog device and all users. However, in real life settings users are spread across location. Thus, instead of using a single value, we have x and y coordinates for both, users, and fog devices. At the beginning of placement algorithm, we calculate Euclidian distance between each user with each fog device and then store all the distances in hastable for fast access. For Objective 4 we have two modifications. One only calculates the sum of all distances for services placed on fog and ignores services that went to cloud. In another modification, we include distance to cloud and, as shown in experiment results, it produces much better results. It is unclear if baseline memetic algorithm account for distance to cloud. In this project, we assume that authors did include distance to cloud in Objective 4.

Objective 5, *F5*: minimize number of active fog devices. We count all fog devices that host at least one service.

Objective 6, *F6*: maximize the number of services with small deadline on fog devices. For this objective, we normalize deadlines for all services and create an inverse of 1. Thus, the smallest deadline value will be close to 1 and the biggest deadline value will be close to 0. We sum all the normalized reversed deadlines for all services that were placed on fog devices.

If service were not placed on fog and instead hosted by cloud, we do not include such deadline in our calculation effectively setting it to 0.

Objective 7, $F7$: maximize the number of fog devices with fastest processor. For this objective for all services in fog we sum the processor speed of fog devices where each service is hosted; thus, prioritizing the processors with fastest speed.

V. Algorithms

A. Baseline algorithms

This section gives an overview of two baseline algorithms. First algorithm used as a baseline is First Fit by RAM, and the second algorithm is Memetic Algorithm with pareto approximation proposed by [2].

First Fit RAM. First-fit by RAM is used as the first baseline algorithm. For this algorithm, we order fog devices by their memory (RAM) availability in ascending order. We iterate through each service, trying to place it on one of the fog devices. This algorithm starts with the fog device with the smallest available memory (RAM) resource availability by checking if all resources, memory (RAM), disk, and core, satisfy the service requirements. This algorithm increases resource utilization by memory since we place service on the fog devices with minimal memory.

First Fit by RAM algorithm

```
 1:  sorted fog devices = sorted by memory fog devices
 2:  for service in services
 3:      for fog device in sorted fog devices
 4:          if fog devices(resources) >= service requirements
 5:              place service on this fog device
 6:              update fog device resource availability
 7:              break
 8:          end
 9:      end for
10:      if service not placed on fog:
11:          place service on cloud
12:      end
13:  end for
```

Memetic baseline. The memetic algorithm belongs to evolutionary algorithms which are inspired by nature. The idea is to generate a population of creatures where each creature is a possible solution. Each generation selects the best creatures based on the fitness function, which consists of multiple objective functions, and the new population is generated. The traditional genetic algorithm is prone to premature convergence leading to the suboptimal final solution. The memetic algorithm introduces a local search step that optimizes each population and prevents premature algorithm convergence.

In their work, Sami [2] uses the memetic algorithm with Pareto set approximation. In each generation, for all creatures, dominant sorting is performed. One solution is dominant over another if all objective function values of the first solutions are at least as good as all objective function values of the second solution, and at least one objective function value improves the solution. This algorithm allows determining the Pareto front, the most promising solutions. For each generation, all promising solutions are stored. At the end of the algorithm, dominant sorting is performed for the last time to find the final Pareto front and choose the best solution of the algorithm. The baseline algorithm uses five objective functions described in the previous section with equal weights. The flow chart is shown on Figure 2 and pseudo code for memetic baseline algorithm shown below.

```
Memetic baseline algorithm
 1:  Check if the problem has a solution
 2:  Initialize set of solutions P0
 3:  P0’ = repair infeasible solutions of P0
 4:  P0’’ = apply local search to solutions of P0’
 5:  Update set of non-dominated solutions Pknown from P0’’
 6:  t = 0
 7:  Pt = P0’’
 8:  while (stopping criterion is not met), do
 9:      Qt = selection of solutions from Pt ∪ Pknown
10:      Qt’ = crossover and mutation of solutions of Qt
11:      Qt’’ = repair infeasible solutions of Qt’
12:      Qt’’’ = apply local search to solutions of Qt’’
13:      increment t
14:      Update set of non-dominated solutions Pknown from Qt’’’
```

```
15:     Pt = fitness selection from Pt ∪ Qt'''
16:     end while
17: return Pareto set approximation Pknown
```

Baseline Local Search

```
1: Probability: Random value between zero and one
2: while there are solutions not verified do
3:     if Probability < 0.5 then
4:         We remove services placed on Hj and run them on Hj' if resources available are enough, and then assign any unselected service on Hj if resources are available after sorting them with priority level
5:     else
6:         We assign all services Si needed to available Hj devices depending on resources requirement, and then we discard all Hj and assign all services Si to new set of volunteers Hj' that can host them
7:     end
8: end while
9: return Set of Optimized Solutions Pt"
```

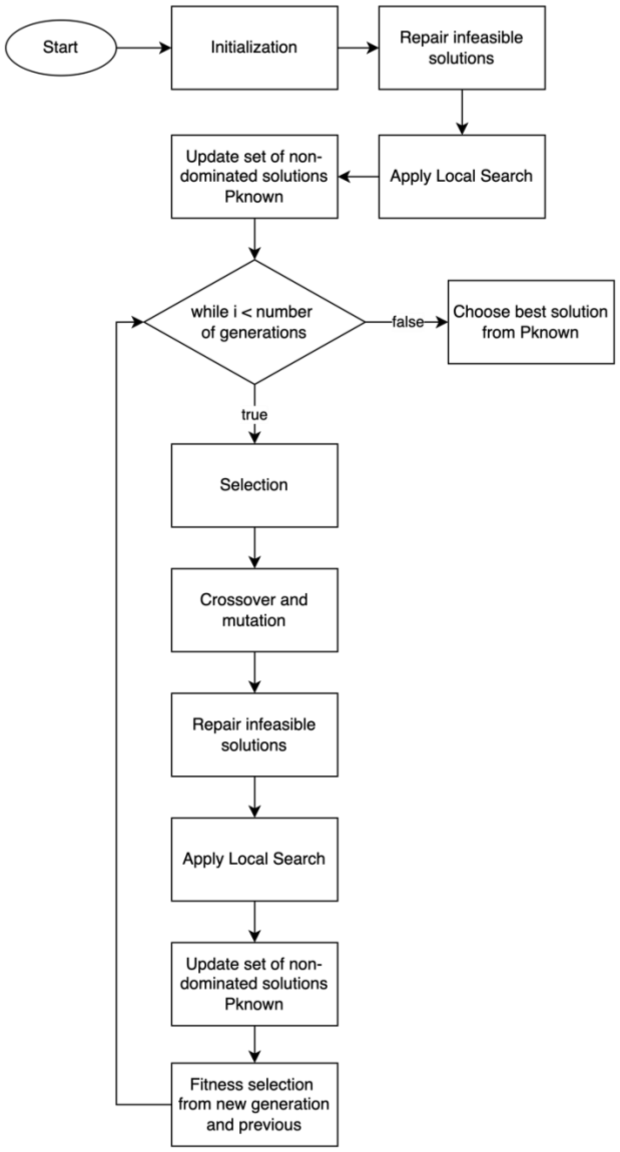
Start
Initialization
Repair infeasible solutions
Apply Local Search
Update set of non-dominated solutions Pknown
while i < number of generations
false
Choose best solution from Pknown
true
Selection
Crossover and mutation
Repair infeasible solutions
Apply Local Search
Update set of non-dominated solutions Pknown
Fitness selection from new generation and previous

Figure 2. Flow chart for Baseline Memetic algorithm.

B. *Memetic Algorithm Modifications*

The following modifications were created and added together in different combinations summarized in Table 2.

Distance from user to fog device. In our architecture, the orchestrator is not topology-aware and only can estimate network propagation based on the distance between user and fog device. The distance to fog devices was set to the same value in [2], which is used as a baseline algorithm. The authors did not take into consideration that users can be in a different location relative to fog devices. We introduce x and y coordinates to express the physical location of fog devices and users. At the beginning of the algorithm, we calculate the Euclidian distance between each user and fog device and store this information in Hashable for efficient runtime access. We also assign x and y coordinates for the centralized cloud to use this distance in objective calculations. Based on the distance between fog devices, the network propagation delay is assigned in a given range. The x and y coordinates are used for all algorithms, including the baseline memetic.

Table 2. Comparison table of different combinations of modifications created for memetic baseline algorithm.

Algorithm	Local search, maximize number of services algorithms	Local search, minimize number of fog devices algorithm	Local search frequency, (local search happen every N generation)	Fitness function includes distance to cloud	Using new objectives F6 and F7	Using ML model to choose the best solution from pareto optimal
Memetic Baseline	old	old	1	✓		
Memetic without Local Search	-	-	-	✓		
Experimental 1	old	old	2	✓		
Experimental 2	new	old	1	✓		
Experimental 3	new	-	1	✓		
Experimental 4	old	old	1			
Experimental 5	new	old	1			
Experimental 6	new	old	2	✓		
Experimental 7	new	old	1	✓	✓	
Experimental 8	old	old	1	✓	✓	
Experimental 9	old	old	1	✓	✓	✓

Local search frequency adjustment. The baseline memetic algorithm is relatively slow. The profiling of python code for baseline memetic algorithm was performed using cProfile to indicate the most inefficient parts of code. The result shows that the main factor affecting time in baseline memetic is the local search performed on every creature in every generation. We created a memetic algorithm without local search to estimate how much time it would take and how it would affect the metrics. Additionally, we introduce a variable that regulates the frequency of how often the local search is performed. For example, by assigning the frequency variable to 2, we perform the local search only every second generation. In experiment analysis, we discuss how it affects the calculation time of the placement algorithm.

New heuristic for local search. The local search consists of two parts – minimization of the number of active fog devices and maximization of services placed on fog. Both parts of local search are applied to all creatures in the population. With the probability of 0.5, the order of applied parts differs. For example, if the generated probability is less than 0.5, firstly, minimization of the number of active fog devices performed first and then maximization of services placed on fog. If the probability is greater than 0.5, maximization of services performed first and then the minimization of fog devices. To improve the algorithm's performance, we proposed our own algorithm for the maximization of services on fog.

In the original maximization of services, the algorithm iterates over services that already have placement on fog devices, trying to place unassigned service to the same fog device. This approach does not consider the distance between the user of the service and the fog device and does not consider idle fog devices.

In our proposed heuristic for maximization of services in fog, we first create a Hash table where the key is a service, and the value is the list of all the fog devices sorted by the distance in ascending order. Since the distance between users and fog devices does not change over time in our architecture, this Hash table is calculated only once at the beginning of the memetic algorithm. When the maximization of services is performed, the algorithm fetches the list of all fog devices from Hash table sorted by distance relative to the user of this service. Algorithm iterates over this list of fog devices starting from closest to the user fog device until it finds one that has enough resources to host this service. The pseudo-code for this heuristic is shown below.

```
Local search heuristic to maximize the number of services in fog
1:  service_to_fog_devices = hashtable where key is service id, value is a sorted by
    distance fog devices
2:  for service in services
3:     if service not assigned to any fog device then
4:        list_of_fog_devices = service_to_fog_devices.get(service id)
5:        for fog_device in list_of_fog_devices
6:           if fog_device resources >= service resource requirements then
7:              assign service to this fog_device
8:              break
9:           end
10:       end for
11:    end
12: end for
```

Introduction of two new objective functions. Since requests cannot wait forever to be processed in the real-world scenario, we introduced a deadline for each service's requests. To make the memetic algorithm aware of deadlines, we add a new objective to the fitness function to tune the population for this objective. This objective function is described in Section 4. It calculates the sum of the normalized inversed deadlines for services placed on fog where the services with the smallest deadlines contribute the most to this objective.

The second new objective function was introduced to improve the total response time. The total response time is when the user's request was emitted until the request was successfully processed. The total response time includes the network latency, the time taken to deliver the request from a user to the fog device, and the time it took to finish the calculation on the fog device. Since we have a heterogeneous environment and each fog device may have the different processing power, we want to account for it as well. For our placement algorithm, we want to choose the fog devices that satisfy resource requirements and have the fastest processors. For each service placed on fog, the objective function will sum the processor instruction per time interval (IPT) value, and our algorithm will try to maximize this value.

The described modifications for different memetic algorithm configurations are summarized in Table 2. The differences and similarities between algorithms are shown on Figure 3.

The memetic baseline is a parent algorithm and other algorithms are modification of the memetic baselisne. We add Memetic without local search, which only differs from baseline, by not performing the local search to compare how it affects the calculation time and how much it worsens the placement results.

The modification Experimental 1 is similar to memetic baseline but performs local search only for every second generation. For this modification, we again want to compare how skipping local search every second generation will affect the calculation time and how much the placement results will be affected.

The modification Experimental 2 is very similar to baseline but uses the new greedy heuristic to find fog devices with the smallest distance for unassigned service. The modification Experimental 6 is a copy of Experimental 2 but performs local search with the new heuristic only for every second generation.

For modification Experimental 3, we use only the new heuristic, ignoring the minimization of active fog devices.

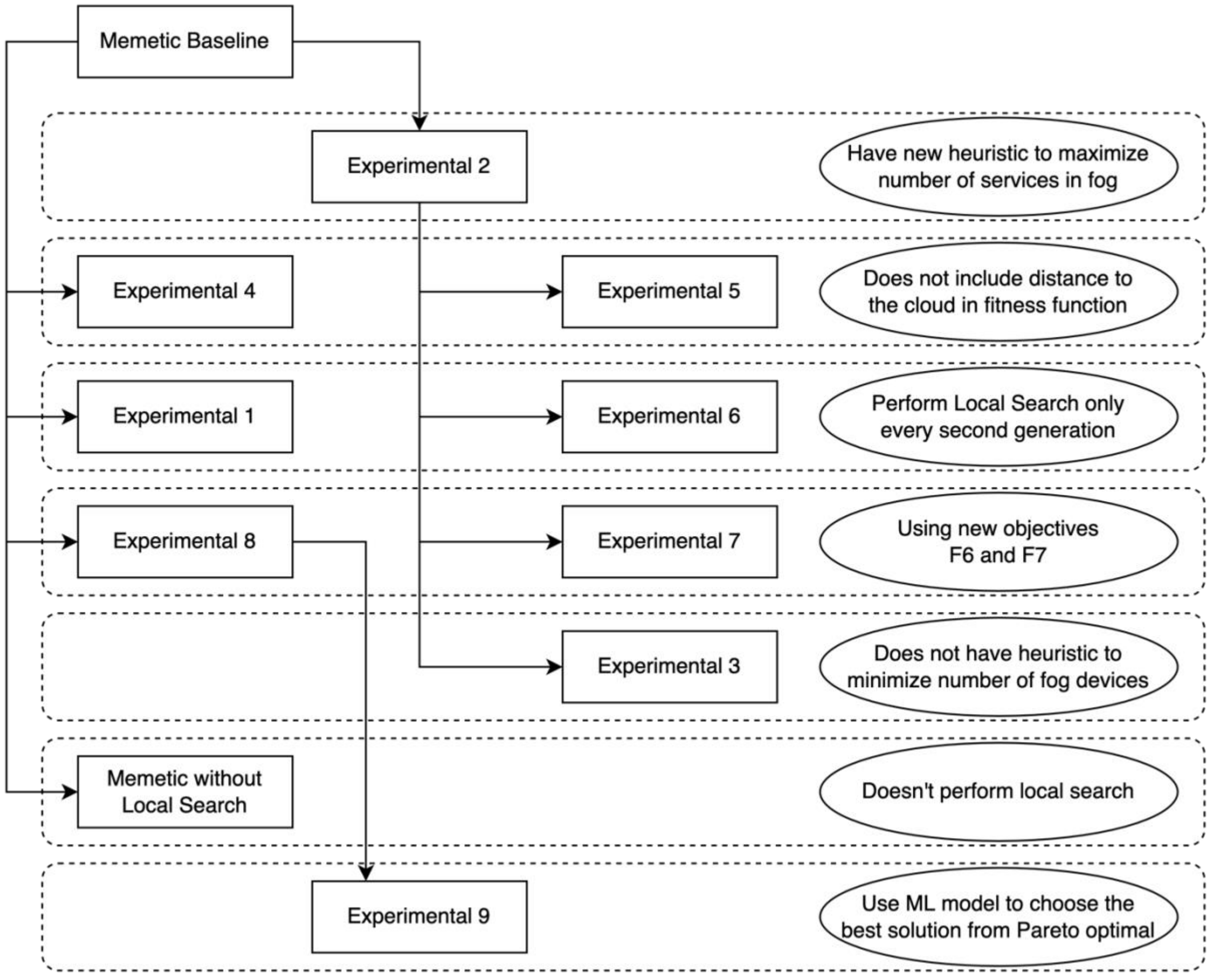
Memetic Baseline
Experimental 2
Have new heuristic to maximize number of services in fog
Experimental 4
Experimental 5
Does not include distance to the cloud in fitness function
Experimental 1
Experimental 6
Perform Local Search only every second generation
Experimental 8
Experimental 7
Using new objectives F6 and F7
Experimental 3
Does not have heuristic to minimize number of fog devices
Memetic without Local Search
Doesn't perform local search
Experimental 9
Use ML model to choose the best solution from Pareto optimal

Figure 3. Comparison of memetic algorithm modifications. Rectangles are placement algorithms. An arrow shows what the base algorithm for each modification is. An ellipse shows the changes introduced in the algorithms enclosed in the dashed box.

Modifications Experimental 4 and Experimental 5 are similar to Memetic Baseline and Experimental 2 with the new heuristic. In these two modifications, we want to explore the importance of adding distance to the cloud for the objective that calculates the distances between user and fog device where the user’s service is placed. For all other modifications, if service were not placed on fog device and instead goes to the cloud, we add a distance to the cloud as a punishment. Since the first objective, F1, already calculates the number of services placed on fog, this punishment may seem redundant. With Experimental 4 and Experimental 5, we do not add distance to the cloud and only try to minimize the distances between users and fog devices inside the fog cluster.

With modification Experimental 7 and Experimental 8, we want to test how the two new objective functions affect the placement algorithms and if they help to reduce the number of requests filed by the deadline. Experimental 7 is similar to Experimental 2, but while Experimental 2 has five objective functions, Experimental 7 has seven objective functions. Experimental 8 is similar to Memetic baseline but has two more objective functions than baseline.

Finally, with modification Experimental 9, we will test the machine learning approach (described below) on the final placement results. Experimental 9 is an extension of Experimental 8, using seven objectives. In addition, for Experimental 9, we use machine learning at the final step of the algorithm to predict which of the solutions from the set Pareto optimal solutions will produce the best result for total response time.

C. Machine Learning Optimization

At the final stage of the memetic algorithm, we have a set of solutions that are Pareto optimal. Since there are multiple solutions, we have to choose one to perform service placement. The straightforward approach is to perform normalization for each objective function. After that, add all objectives that we want to maximize and subtract all objectives that we want to minimize. Finally, compare the final value for each solution picking the solution with the biggest value.

However, this approach may not be the best since the objectives often contradict each other or overshadow each other. One way to overcome it is to tune the weights for each objective function to indicate which objectives are of the most importance. However, we have to adjust it manually, and the dependency between different objectives may not be straightforward. Alternatively, we can use machine learning to learn which values of the objective functions are most desirable to optimize a particular metric. Thus, we will train a machine-learning algorithm to predict the value for the metric that we want to optimize.

The inputs for the prediction will be values for each objective function, the number of services in the fog cluster, and a number of fog devices. The output is the predicted metric value that we want to optimize.

In this work, we optimize the average total response time, the time that takes a request to be sent over the network and processed in a fog device. Based on the prediction from the machine learning model, we can choose the final solution among Pareto optimal solutions. This project uses a machine learning approach with seven objective functions. We use Experimental 8 to generate a dataset for different values of seven objective functions. Additionally, the dataset will include the basic architecture information, such as the number of fog devices and the number of services to be placed. The output is calculated from simulation results, removing the outliers. We use a neural network with five hidden layers for the machine-learning model.

VI. SIMULATION

The baseline memetic algorithm and its modifications were implemented using Python 3.6 and the NumPy library. The placement calculation and simulation were parallelized using the python multiprocessing library. It allowed using all cores to run experiments which helped to obtain results in a reasonable time.

For placement calculations and simulations, we use AWS EC2 c6i.32xlarge with 256 GiB memory, 200 GiB storage, and 128 CPUs. This instance allowed to calculate of 128 experiments at the same time.

All source code available on GitHub https://github.com/msurmenok/master-project.

Simulator framework. We use Yet Another Fog Simulator (YAFS) to simulate the fog environment and calculate the efficiency of the proposed algorithms. YAFS is a discrete-event simulator using complex network theory. It is written in python and has good documentation. YAFS simulator allows simulating initial and dynamic placement and dynamic events in the system, such as user movement. For each simulation run, the simulator logs all simulation results to two CSV files; one file describes the lifespan for each request, including the time when the request was emitted, when the fog device accepted it, and when the request was processed. Another file describes the transmission events between network links. As the authors say, “there are no magic hidden variables” [14]. All data is fine-grained and gives the flexibility to create custom metrics.

Experiment setup. A single experiment generates a single network setup, set of applications with all the requirements, user distribution, and placement for each algorithm for a given network and applications. It allows testing placement on the same network settings. All this data is saved as a JSON file and loaded to simulate each placement.

In this project, we do 100 experiments for two scenarios. The first scenario has 20 fog devices and 50 services, and the second scenario has 40 fog devices and 100 services to be placed. These scenarios are described in Table 3. The same scenarios are used for baseline memetic [2]. We run simulation with the same network size but different network and application configurations 100 times for each experiment configuration.

For memetic algorithms, we use 1000 generations with a population of 100 creatures similar to [2].

Table 3. Experiment scenarios.

	Number of fog devices	Number of services	Number of gateways	Number of experiments
configuration 1	20	50	5	100
configuration 2	40	100	10	100

We generate network and application configurations randomly, using values similarly to [15]. In this paper, the authors test their rank-based placement algorithm for fog using the YAFS simulator. The value ranges are shown in Table 4 and Table 5. The network is generated using python library *networkx* that can produce realistic networks. To obtain x and y coordinates, we map network representation to the 2d plane using *networkx* library tools. To determine which nodes on the network should be gateway devices, we perform network centrality calculations. It finds the most popular paths and assigns a rank to each node. The network nodes with the lowest rank are considered edge devices, so we assign them to be gateway devices that cannot host services but emit requests on behalf of users. The network node with the highest centrality rank is considered the one with the highest traffic. We assume that this is the gateway for a centralized cloud. To emulate a centralized cloud, we create one more node in the network and set all values following the Table 5 values for the cloud.

We consider five network and application settings. For all five configurations we perform the same number of simulations described in Table 3. List of five network and application settings configurations:

- Average network and applications configuration
- Configuration with requests above average
- Configuration with requests below average
- Configuration with fog devices' resources above average
- Configuration with fog devices' resources below average

Table 4 and Table 5 shows network and application settings for average network and applications configuration. The next four are slightly differs from the average settings.

Table 4. Application settings.

Parameter	Value (min – max)
memory (units)	1 – 5
storage (units)	10 – 50
processor (units)	0.1 – 0.5
deadline (ms)	2000 – 20,000
execution (instr/req)	20000 – 60000
message size (bytes)	1500000 – 4500000
priority	0 – 1
user request ratio	1/1000 – 1/200

Configurations for requests above and below average are shown on Table 6 and Table 7 respectively. The only difference from average configuration is the user request ratio.

Configurations for fog devices' resources above and below average have the same applications settings as average configuration. These two configurations only differs from the average configuration by minimum and maximum values for memory, storage, processor, and IPT. The changes are shown in Table 8 and Table 9.

For the configuration with average network and applications setting we perform experiments with all algorithms described in Table 2. For the last four configurations, we use only memetic baseline algorithm and three best performing algorithms from average configuration, Experimental 2, Experimental 6, Experimental 8, and Experimental 9.

Table 5. Fog devices and cloud settings.

Parameter	Fog devices, value (min -max)	Cloud, value
memory (units)	10 – 25	99999999999
storage (units)	20 – 200	99999999999
processor (units)	0.2 – 2.0	99999999999
IPT (instr / ms)	500 – 1000	10000
bandwidth (bytes/ ms)	75000	125000
propagation delay (ms)	2 – 10	500
x (meters)	0 – 1000	18200
y (meters)	0 – 1000	18200
time availability (ms)	100000 – 2000000	99999999999

Table 6. Application settings for configuration with requests frequency above average.

Parameter	Value (min – max)
memory (units)	1 - 5
storage (units)	10 - 50
processor (units)	0.1 - 0.5
deadline (ms)	2000 - 20,000
execution (instr/req)	20000 - 60000
message size (bytes)	1500000 - 4500000
priority	0 - 1
user request ratio	1/600 - 1/200

Table 7. Application settings for configuration with requests frequency below average.

Parameter	Value (min - max)
memory (units)	1 - 5
storage (units)	10 - 50
processor (units)	0.1 - 0.5
deadline (ms)	2000 - 20,000
execution (instr/req)	20000 - 60000
message size (bytes)	1500000 - 4500000
priority	0 - 1
user request ratio	1/1000 - 1/600

Table 8. Fog devices settings for configuration with fog devices' resources above average.

Parameter	Fog devices, value (min -max)
memory (units)	17.5 - 25
storage (units)	110 - 200
processor (units)	1.1 - 2.0
IPT (instr / ms)	750 - 1000
bandwidth (bytes/ ms)	75000
propagation delay (ms)	2 - 10
x (meters)	0 - 1000
y (meters)	0 - 1000
time availability (ms)	100000 - 2000000

Table 9. Fog devices settings for configuration with fog devices' resources below average.

Parameter	Fog devices, value (min -max)
memory (units)	10 - 17.5
storage (units)	20 - 110
processor (units)	0.2 - 1.1
IPT (instr / ms)	500 - 750
bandwidth (bytes/ ms)	75000
propagation delay (ms)	2 - 10
x (meters)	0 - 1000
y (meters)	0 - 1000
time availability (ms)	100000 – 2000000

Machine learning model. We use the scikit-learn library to create a neural network consisting of five fully connected hidden layers. We use a dataset with approximately 110,000 records from simulation results generated for memetic algorithm Experimental 8. For data generation, we use both placements from Pareto optimal and placements not included in Pareto optimal to have variety in the dataset.

We use the following values as features:

- number of fog devices (either 20 or 40)
- number of services to be placed (either 50 or 100)
- the values for seven objective functions

The average total response time calculated by the YAFS simulator is used as output.

75% of the dataset is used for training, and 25% is used for testing. We use a scikit-learn standard scaler for normalization that subtracts mean and then scales to unit variants, thus centering data around 0.

To measure the accuracy of the trained model, we use the built-in R2 score provided by the scikit-learn library, which shows the coefficient of determination. For our settings, we achieved a score of 0.9448, where 1.0 is the maximum possible value.

To use the trained model in simulation, we serialized the machine learning pipeline, including the standard scaler, and then loaded it for Experimental 9 during placement calculation.

VII. RESULTS AND ANALYSIS

In this section, we will discuss the metrics used for algorithm comparison and discuss simulation results. For each simulation run, YAFS produces a CSV file with information about each request. We use python with NumPy and pandas libraries to calculate metrics and save the results on disk.

A. Metrics.

In this project we aim to optimize:

- Minimize the average total response time – time when request was emitted from user until the request was successfully processed, calculated for all requests that finished within a deadline.
- Minimize the average total response time for important services – time when request for service with maximum priority was send from the user until it was successfully processed, calculated for all requests for important services that finished within a deadline.
- Minimize the average percentage of failed requests – calculated as number of failed requests divided by the total number of requests. The average of all experiments for the same placement is taken and multiplied by 100%.
- Minimize the average percentage of failed requests for important services – similar to the above, but calculate the number of failed requests for important services divided by the total number of requests for important services. The average of all experiments for the same placement algorithm is taken and converted to percent.
- Minimize number of fog devices used – to minimize the cost and energy consumption, we want to minimize the number of fog devices that host at least one service.

- Minimize calculation time – memetic algorithms take a lot of time to complete, we want to find how much optimizations such as skipping local search every second generation affect the calculation time.

B. Results and Discussion for Configuration with Average Network and Applications Settings

All the calculated metrics are shown in Table 10 for scenario with 20 fog devices and 50 services and in Table 11 for scenario with 40 fog devices and 100 services.

Table 10. Results of 100 experiments for scenario with 20 fog devices and 50 services.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority	Number of fog devices used	Average calculation time	Average number of services placed on fog devices
FirstFitRAM	82.9206	0.5987	350.1489	0.6432	18.7600	0.0013	48.3400
Memetic Baseline	76.2812	0.2696	130.6646	0.1528	16.5900	2490.6920	49.4600
Without LC	78.3456	0.3590	150.5299	0.2018	17.7200	1443.0573	49.2400
Experimental 1	76.6408	0.2418	115.2661	0.1217	16.6800	2378.2789	49.4700
Experimental 2	75.9738	0.2861	145.7647	0.2334	15.7800	2794.9338	49.5000
Experimental 3	76.1980	0.2886	146.2898	0.1861	16.6500	3007.4164	49.3200
Experimental 4	83.6501	0.4940	154.1095	0.2002	16.4400	4854.0869	48.7900
Experimental 5	76.8831	0.3433	172.2013	0.2749	15.9400	4424.7323	49.2200
Experimental 6	74.6390	0.2577	144.8497	0.2119	15.9300	1803.0935	49.4600
Experimental 7	83.8878	0.6021	337.8629	0.6678	18.3700	4803.6266	48.5400
Experimental 8	73.5828	0.1681	130.0655	0.1425	16.8700	4751.9233	49.6000
Experimental 9	72.8178	0.1719	133.4792	0.1743	18.9700	5025.9895	49.6000

Table 11. Results for 100 experiments for scenario with 40 fog devices and 100 services.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority	Number of fog devices used	Average calculation time	Average number of services placed on fog devices
FirstFitRAM	76.5162	0.3447	251.1665	0.3961	38.5300	0.0033	97.4200
Memetic Baseline	75.8480	0.1713	120.3254	0.1219	35.7100	7443.5925	99.1800
Without LC	76.8407	0.2436	148.4836	0.1906	37.3800	1698.2685	98.6900
Experimental 1	76.2805	0.1740	134.2103	0.1541	35.6900	5132.9300	99.0300
Experimental 2	75.3157	0.1526	123.3200	0.1218	34.6500	8340.1114	99.2100
Experimental 3	77.7016	0.2434	157.0078	0.2106	35.7000	8195.3604	98.6600
Experimental 4	79.1110	0.2850	160.0949	0.2058	35.4700	12071.6703	98.3700
Experimental 5	76.4428	0.2159	153.9512	0.1831	34.7000	11361.9545	98.7100
Experimental 6	75.2248	0.1535	148.1194	0.1866	34.6800	4987.6597	99.2400
Experimental 7	78.3993	0.3904	234.2334	0.3815	37.7500	10690.2836	97.6800
Experimental 8	75.9687	0.1461	116.7000	0.1023	36.0000	10697.3509	99.3200
Experimental 9	73.9590	0.1086	134.8411	0.1514	37.3400	10128.9817	99.4200

The average calculation time for all memetic algorithms for both scenarios is shown on Figure 4. The comparison of average total response time shown on Figure 5. The comparison of percentage of failed requests are shown in Figure 6. For services with the highest priority, the average total response time and percentage of failed requests are shown in Figure 7 and Figure 8 respectively. Figure 9 and Figure 10 show the average number of fog devices that host at least one service.

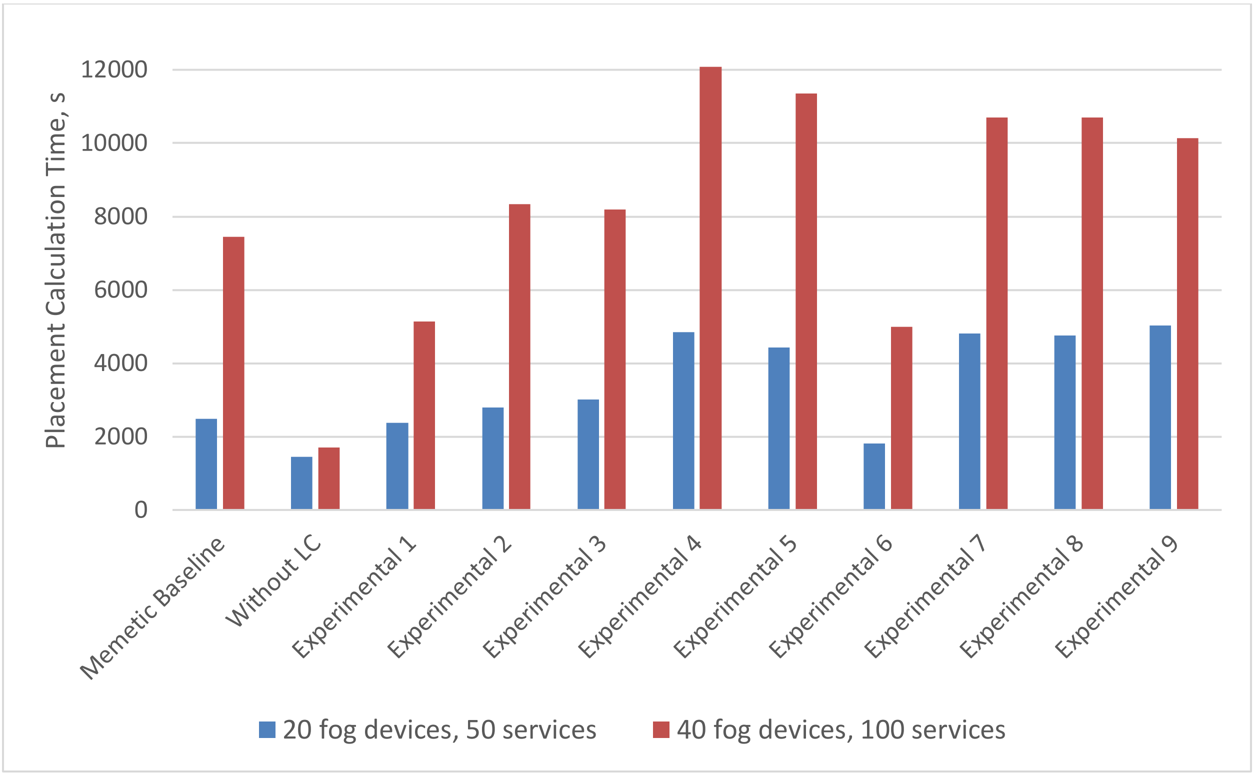

Figure 4. Average calculation time for modifications of memetic algorithm.

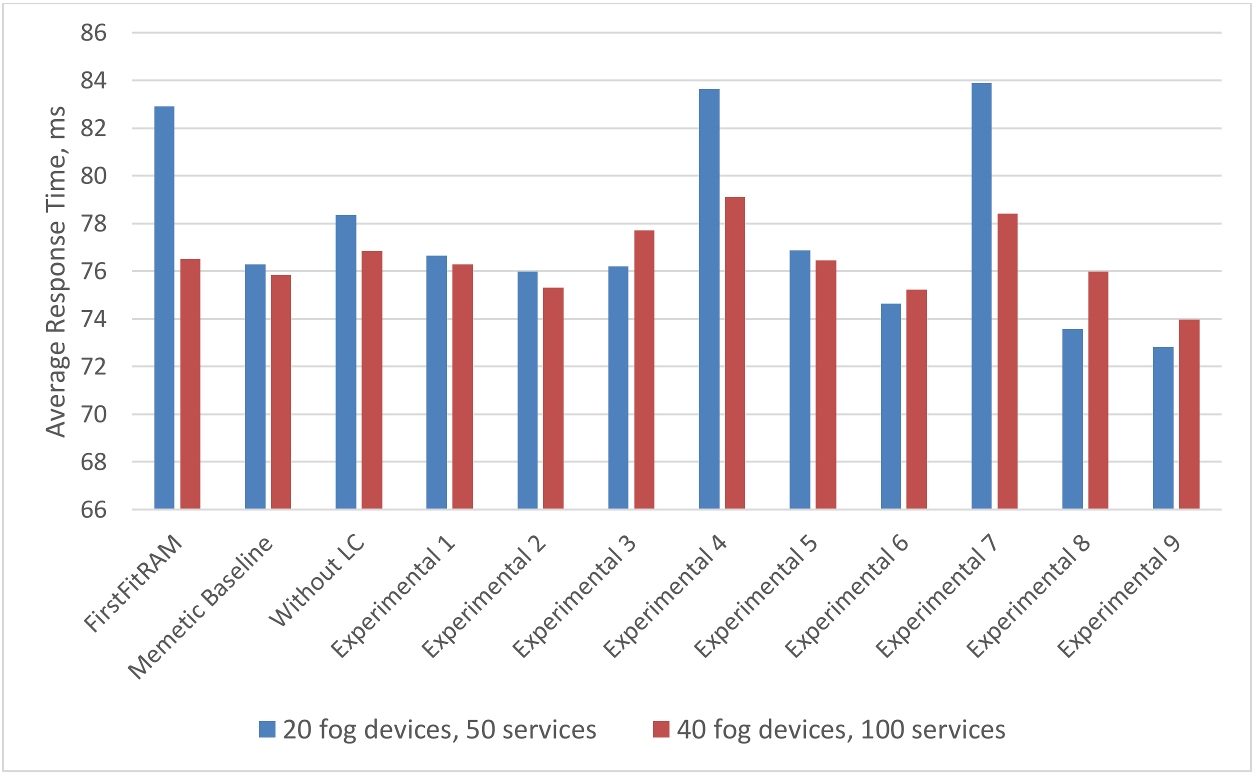

Figure 5. Average total response time.

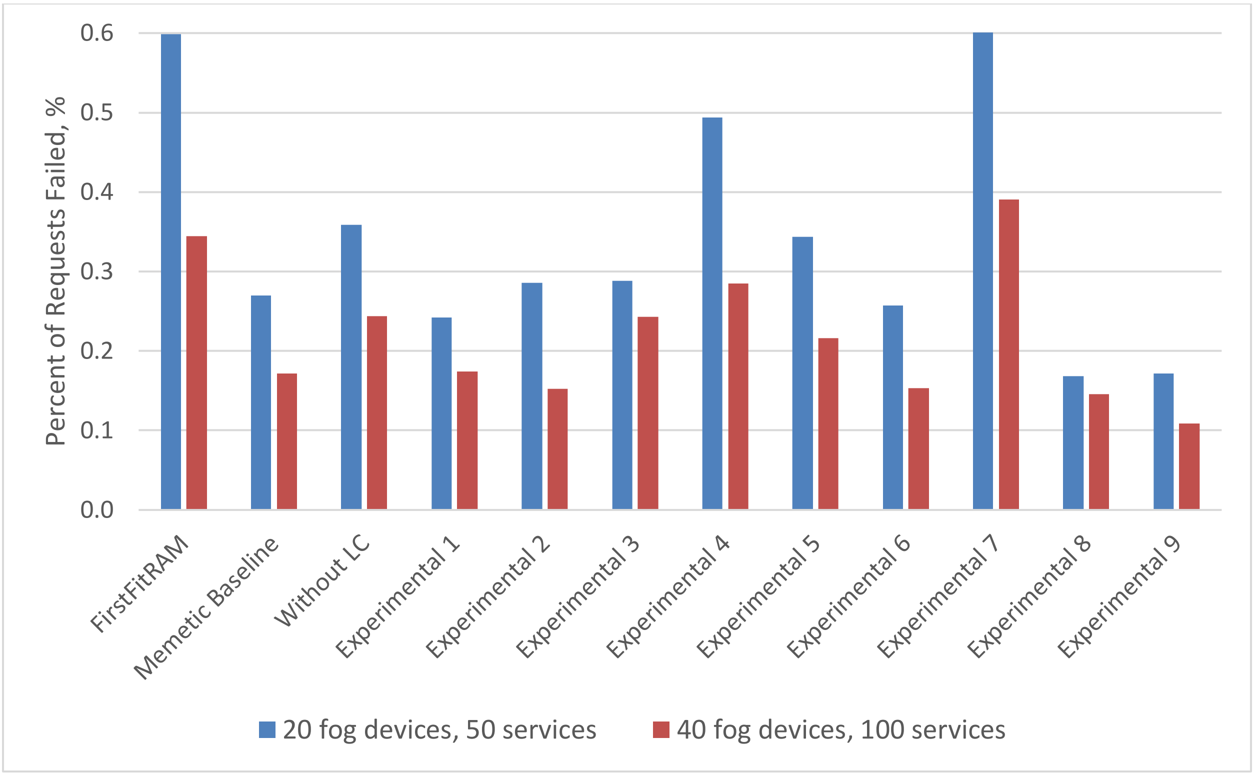

Figure 6. Average percent of failed requests.

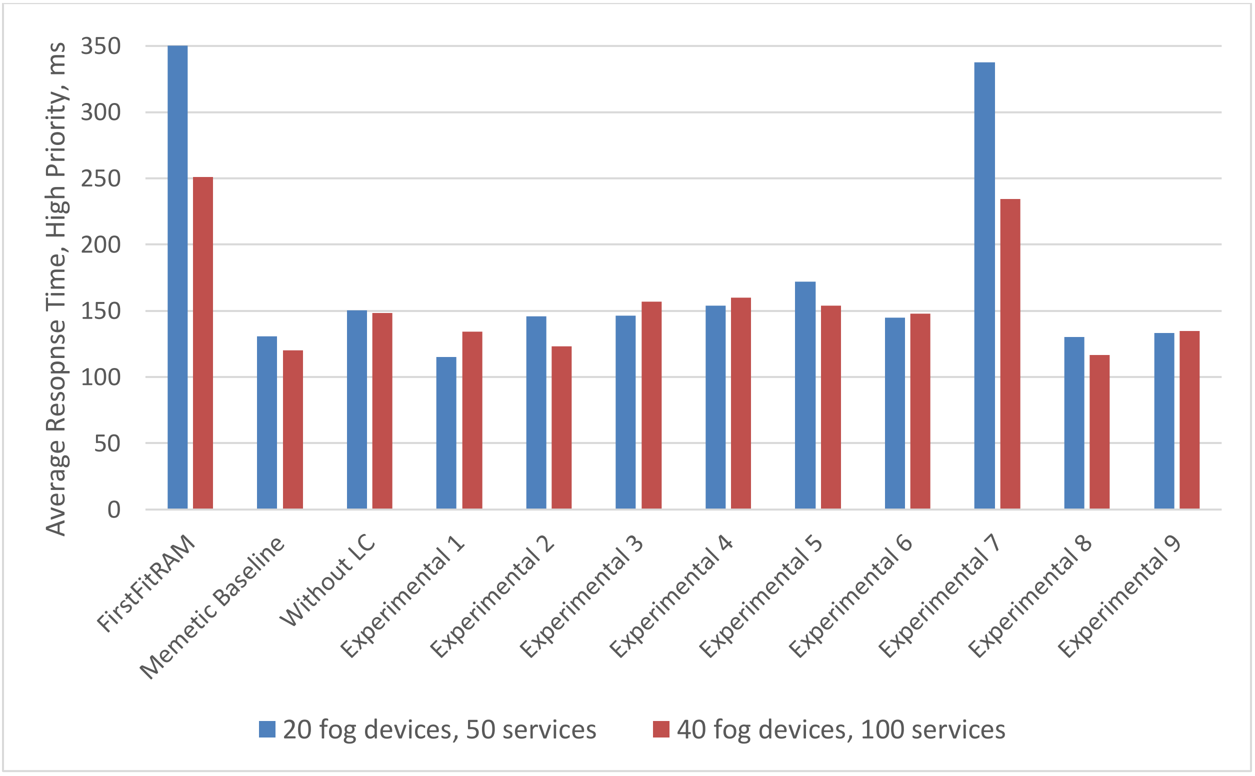

Figure 7. Average total response time for services with high priority.

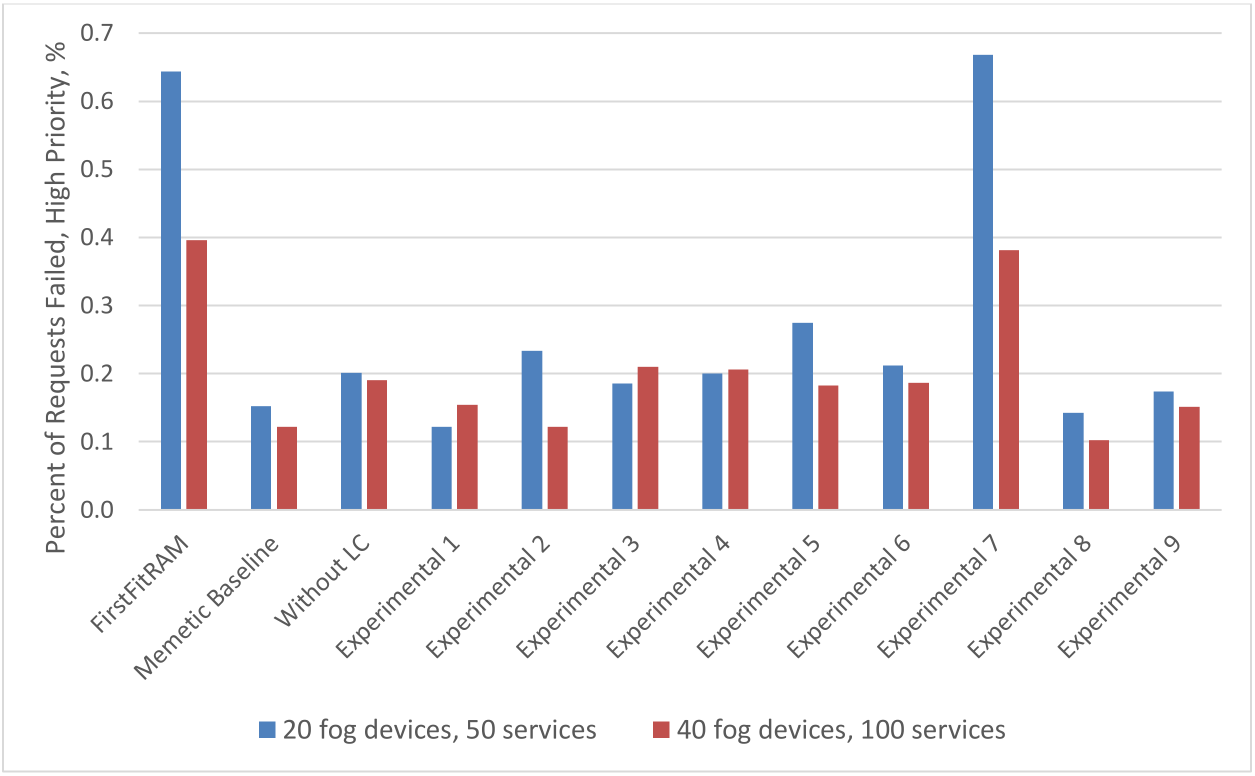

Figure 8. Average percent of failed requests for services with high priority

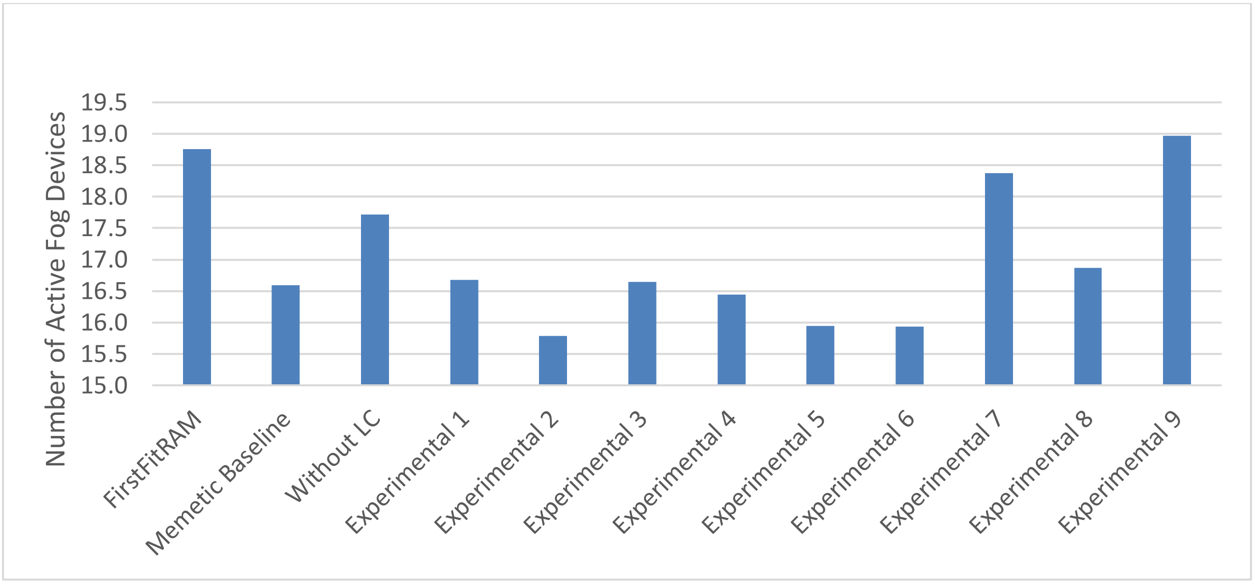

Figure 9. Average number of active devices for scenario with 20 fog devices and 50 services.

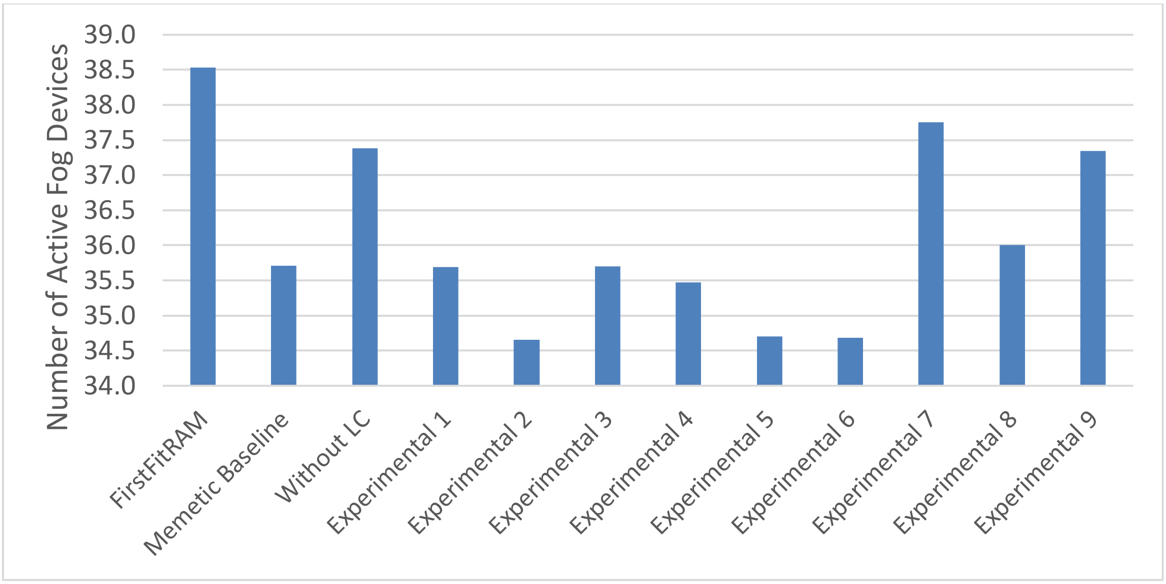

Figure 10. Average number of active devices for scenario with 40 fog devices and 100 services.

Memetic modification Experimental 2 uses a combination of a new heuristic with minimization of a number of fog devices from old local search. The subset of results for Experimental 2 and baseline algorithms is shown on Table 12 and Table 13.

Table 12. Comparison of results for Experimental 2 with baseline algorithms of 100 experiments for scenario with 20 fog devices and 50 services. Average configuration.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority	Number of fog devices used	Average number of services placed on fog devices
FirstFitRAM	82.9206	0.5987	350.1489	0.6432	18.7600	48.3400
Memetic Baseline	76.2812	0.2696	130.6646	0.1528	16.5900	49.4600
Experimental 2	75.9738	0.2861	145.7647	0.2334	15.7800	49.5000

The results show that Experimental 2 improves the average total response time compared to Memetic Baseline [2]. It can be explained by the number of services placed on fog devices where Experimental 2 could place more services on average, thus reducing the number of requests sent to the cloud for processing.

Table 13. Comparison of results for Experimental 2 with baseline algorithms of 100 experiments for scenario with 40 fog devices and 100 services. Average configuration.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority	Number of fog devices used	Average number of services placed on fog devices
FirstFitRAM	76.5162	0.3447	251.1665	0.3961	38.5300	97.4200
Memetic Baseline	75.8480	0.1713	120.3254	0.1219	35.7100	99.1800
Experimental 2	75.3157	0.1526	123.3200	0.1218	34.6500	99.2100

Although Experimental 2 uses a greedy heuristic, it could employ fewer fog devices for placement. The network setup can explain this—all users are connected to a small number of gateways that send requests on the users' behalf. When we calculate the distance between fog devices and users, we use the coordinates of users' gateway devices. Thus, for users connected to the same gateway, the closest fog devices will be in the same order. Therefore, Experimental 2 will explore the closest fog devices in the same order for multiple users allowing for packing services more tightly.

Even though Experimental 2 has more services placed on fog devices, it performs worse for services with high priority. This can be explained by using all objective functions with equal weight when the improvement in one objective overshadows other objective functions. The difference in performance is greater in a small network setup with 20 fog devices and 50 services. In our simulation, we use only the described devices for network communication. Therefore, fewer fog devices lead to fewer paths to send requests and consequently to increased network congestion.

Overall, from Table 12 and Table 13, we can see the difference in performance between First Fit and both memetics. Thus, although First Fit placement is calculated quickly compared to the memetic algorithm, it cannot achieve the same performance. For services with high priority, baseline memetic algorithm outperforms First Fit percentage of failed requests by up to 76.56% and average total response time by up to 62.17%. Thus, using memetic algorithms can be justified when we expect extended time usage of services and the number of failed requests and total response time for services with high priority is more important than initial placement time.

Experimental 6 differs from Experimental 2 only by performing a local search for every second generation, reducing the overall calculation time. The subset of results for Experimental 6 and baseline algorithms is shown in Table 14 and Table 15**.** We can see that Experimental 6 has similar or slightly worse results compared to its parent algorithm, Experimental 2 and Memetic Baseline. Since Experimental 6 performs local search only every second generation, which is shown by profiler bottleneck for memetic algorithms, Experimental 6 has a shorter calculation time. The experiments show that Experimental 6 reduces calculation time by up to 33% compared to Memetic Baseline and up to 40% compared to Experimental 2. Experimental 6 can be a feasible solution for settings where we need to calculate placement while still having near-optimal performance.

Table 14. Comparison of results for Experimental 6 with baseline algorithms of 100 experiments for scenario with 20 fog devices and 50 services. Average configuration.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority	Average calculation time
FirstFitRAM	82.9206	0.5987	350.1489	0.6432	0.0013
Memetic Baseline	76.2812	0.2696	130.6646	0.1528	2490.6920
Experimental 2	75.9738	0.2861	145.7647	0.2334	2794.9338
Experimental 6	74.6390	0.2577	144.8497	0.2119	1803.0935

Table 15. Comparison of results for Experimental 6 with baseline algorithms of 100 experiments for scenario with 40 fog devices and 100 services. Average configuration.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority	Average calculation time
FirstFitRAM	76.5162	0.3447	251.1665	0.3961	0.0033
Memetic Baseline	75.8480	0.1713	120.3254	0.1219	7443.5925
Experimental 2	75.3157	0.1526	123.3200	0.1218	8340.1114
Experimental 6	75.2248	0.1535	148.1194	0.1866	4987.6597

The memetic modification Experimental 3 uses only a new greedy heuristic searching for fog devices nearby without minimizing the number of fog devices in its local search. The subset of results for Experimental 6 and baseline algorithms is shown on Table 16 and Table 17. Compared to baseline

Table 16. Comparison of results for Experimental 3 with baseline algorithms of 100 experiments for scenario with 20 fog devices and 50 services. Average configuration.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority
FirstFitRAM	82.9206	0.5987	350.1489	0.6432
Memetic Baseline	76.2812	0.2696	130.6646	0.1528
Experimental 3	76.1980	0.2886	146.2898	0.1861

Table 17. Comparison of results for Experimental 3 with baseline algorithms of 100 experiments for scenario with 40 fog devices and 100 services. Average configuration.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority
FirstFitRAM	76.5162	0.3447	251.1665	0.3961
Memetic Baseline	75.8480	0.1713	120.3254	0.1219
Experimental 3	77.7016	0.2434	157.0078	0.2106

Memetic, Experimental 3 does not show any improvements comparing to Memetic Baseline from [2] while still producing better results comparing to First Fit. The failure to outperform the baseline Memetic can be explained by not minimizing the number of fog devices during the local search. In such a case, only the maximization of placed services is performed using the new greedy algorithm. Without an attempt to pack services tightly to the same fog devices, the maximization algorithm makes an inefficient placement on fogs near the user and reducing utilization efficiency.

The memetic modifications Experimental 4 and Experimental 5 are similar to baseline Memetic and Experimental 2. The only difference is that Experimental 4 and Experimental 5 do not use distance to cloud when calculating objectives that minimize the distance. The subset of results for Experimental 4, Experimental 5 and baseline algorithms is shown on Table 18 and Table 19.

Table 18. Comparison of results for Experimental 4 and Experimental 5 with baseline algorithms of 100 experiments for scenario with 20 fog devices and 50 services. Average configuration.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority
FirstFitRAM	82.9206	0.5987	350.1489	0.6432
Memetic Baseline	76.2812	0.2696	130.6646	0.1528
Experimental 4	83.6501	0.4940	154.1095	0.2002
Experimental 5	76.8831	0.3433	172.2013	0.2749

Table 19. Comparison of results for Experimental 4 and Experimental 5 with baseline algorithms of 100 experiments for scenario with 40 fog devices and 100 services. Average configuration.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority
FirstFitRAM	76.5162	0.3447	251.1665	0.3961
Memetic Baseline	75.8480	0.1713	120.3254	0.1219
Experimental 4	79.1110	0.2850	160.0949	0.2058
Experimental 5	76.4428	0.2159	153.9512	0.1831

Experimental 4 and Experimental 5 show worse performance than baseline Memetic. Although using the distance to the cloud may seem redundant, since we already maximize the number of services in fog in the first objective, algorithms Experimental 4 and Experimental 5, prove that we should consider network latency related to the significant distance to the cloud in another objective function. This demonstrates that the algorithm performs better when it is not only aware of the number of services sent to the centralized cloud but also how far away requests need to be sent. Since we use distance as an approximation of network latency, calculating distance to the cloud can help the algorithm learn how such placement to the cloud may negatively affect network latency.

Memetic modification Experimental 7 is similar to Experimental 2, and Experimental 8 is similar to baseline Memetic. Experimental 7 and Experimental 8 have two newly created objective functions that prioritize services by the deadline and prioritize fog devices with the fastest processors. Thus, both Experimental 7 and Experimental 8 have seven objective functions, while the rest of the modifications have five objective functions. The subset of results for Experimental 7, Experimental 8 and baseline algorithms is shown on Table 20 and Table 21.

Table 20. Comparison of results for Experimental 7 and Experimental 8 with baseline algorithms of 100 experiments for scenario with 20 fog devices and 50 services. Average configuration.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority	Number of fog devices used	Average number of services placed on fog devices
FirstFitRAM	82.9206	0.5987	350.1489	0.6432	18.7600	48.3400
Memetic Baseline	76.2812	0.2696	130.6646	0.1528	16.5900	49.4600
Experimental 2	75.9738	0.2861	145.7647	0.2334	15.7800	49.5000
Experimental 7	83.8878	0.6021	337.8629	0.6678	18.3700	48.5400
Experimental 8	73.5828	0.1681	130.0655	0.1425	16.8700	49.6000

Table 21. Comparison of results for Experimental 4 and Experimental 5 with baseline algorithms of 100 experiments for scenario with 40 fog devices and 100 services. Average configuration.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority	Number of fog devices used	Average number of services placed on fog devices
FirstFitRAM	76.5162	0.3447	251.1665	0.3961	38.5300	97.4200
Memetic Baseline	75.8480	0.1713	120.3254	0.1219	35.7100	99.1800
Experimental 2	75.3157	0.1526	123.3200	0.1218	34.6500	99.2100
Experimental 7	78.3993	0.3904	234.2334	0.3815	37.7500	97.6800
Experimental 8	75.9687	0.1461	116.7000	0.1023	36.0000	99.3200

Experimental 7 shows the worst performance among all the memetic algorithms, including baseline. It has one of the highest averages for the average number of active fog devices and one of the lowest number of services placed on fog devices. The combination of local search looking for fog devices nearby with objective function trying to put the services with the smallest deadline converges to a suboptimal solution.

Experimental 8, on the other hand, is one of the best performing algorithms reducing the number of failed requests for all requests and for requests with higher priority due to prioritizing services with the smallest deadlines. Compared with Memetic Baseline, Experimental 8 improves the percentage of failed requests by up to 37.66% and the percentage of failed requests for services with the highest priority by up to 16.08%. The introduction of two new objectives positively affected the memetic algorithm with local search inherited from baseline Memetic. However, it increases the number of fog devices used for placement to provide enough resources for services with the smallest deadlines.

The subset of results for Experimental 9, its parent algorithm Experimental 9 and baseline algorithms is shown on Table 22 and Table 23.

Table 22. Comparison of results for Experimental 9 with Experimental 8 and baseline algorithms of 100 experiments for scenario with 20 fog devices and 50 services. Average configuration.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority
FirstFitRAM	82.9206	0.5987	350.1489	0.6432
Memetic Baseline	76.2812	0.2696	130.6646	0.1528
Experimental 8	73.5828	0.1681	130.0655	0.1425
Experimental 9	72.8178	0.1719	133.4792	0.1743

Table 23. Comparison of results for Experimental 9 with Experimental 8 and baseline algorithms of 100 experiments for scenario with 40 fog devices and 100 services. Average configuration.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority
FirstFitRAM	76.5162	0.3447	251.1665	0.3961
Memetic Baseline	75.8480	0.1713	120.3254	0.1219
Experimental 8	75.9687	0.1461	116.7000	0.1023
Experimental 9	73.9590	0.1086	134.8411	0.1514

Experimental 9 is an extension of Experimental 8 and only differs by using a machine learning model to choose the solution from Pareto optimal instead of combining results for all objective functions with the same weights. For Experimental 9, we use the ml model to predict the average total response time for all Pareto set solutions, picking the solution with the smallest predicted value. This approach improves the average total response time compared to parent algorithm Experimental 8 by up to 2.65% and comparing to Memetic Baseline by up to 4.54%. Since this algorithm mainly focuses on optimizing the average total response time, the rest of the metrics show worse results than Memetic Baseline but are still acceptable compared to First-Fit. This approach helps optimize and prioritize an average total response time choosing the best solution among Pareto optimal and can be applied to other metrics.

To compare the effect of local search on computational time, we performed experiments with Memetic without local search altogether and two modifications, Experimental 1 and Experimental 6, that perform a local search only every second time. From Figure 11, we can see the comparison of time to calculate placement for different modifications of the memetic algorithm. It illustrates that baseline memetic without local search was calculated almost three times faster than the original local search for both scenarios. However, it severely degrades the average total response time and other metrics. On the other hand, skipping the local search for every second generation shows a 31.04% improvement for Experimental 1 compared to the Memetic Baseline for the scenario with 40 fog devices and 100 services. Experimental 6 differs from Experimental 2 only by performing a local search for every second generation. The scenario with 20 fog devices and 50 services shows the calculation speed up by 35.49% and 40.19% for 40 fog devices and 100 services. While omitting local search altogether has a significant negative effect on other metrics, such as average total response time, Experimental 1 and Experimental 6 produce results close to their versions with local search for every second generation. Thus, algorithms with the local search performed only every second generation may be a viable solution to reduce calculation time while providing optimal or near-optimal placement.

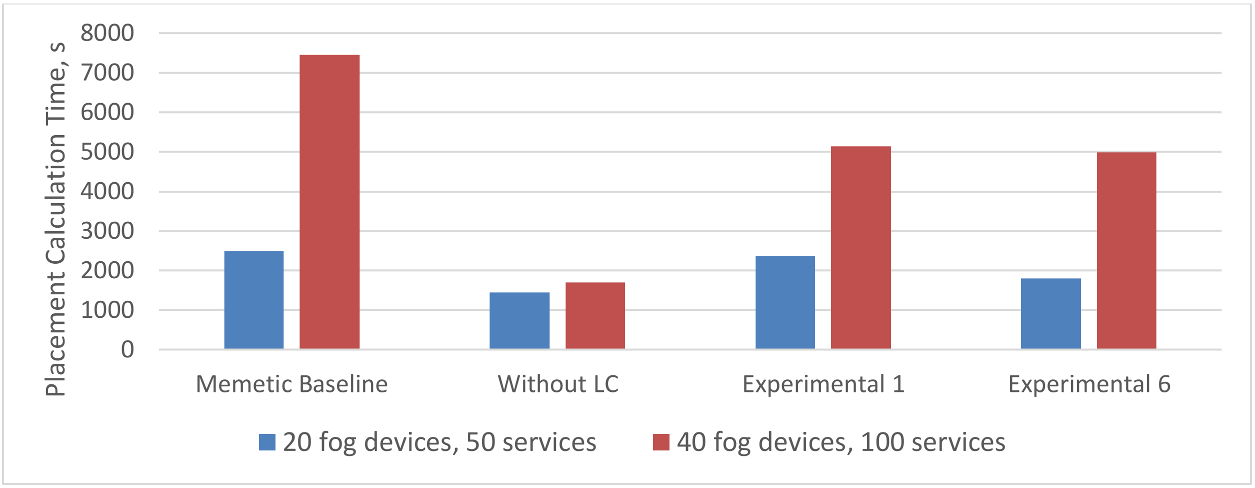

Figure 11. Comparison of average calculation time for memetic baseline, memetic without local search, and two memetic algorithms that perform local search every second generation.

To sum, up Experimental 2, Experimental 6, Experimental 8, and Experimental 9 show the best performance among other placement algorithms. Experimental 9 is the best by average total response time. Experimental 8 is good in average total response and percent of failed requests for all serviced and services with the highest priority. However, Experimental 8 uses more fog devices on average compared to Experimental 2 and Experimental 6. It also has one of the longest calculations, exceeding even baseline Memetic. If the calculation could be parallelized or the prolonged usage of placed services is assumed, Experimental 8 and Experimental 9 are the best algorithm to choose. Experimental 2 has a good performance in terms of total response time and percent of failed requests. It can also place services on the smallest number of fog devices compared to other algorithms. This memetic modification may be helpful if the energy consumption and cost of renting fog devices are important. Finally, Experimental 6 has a very good performance as well. It has one of the smallest average number of fog devices used. This algorithm takes approximately 30% less time to calculate placement than other memetic algorithms while producing excellent results for services with the highest priority.

C. Results and Discussion for Configuration with Different Requests Frequency

This subsection will discuss results for two simulation variations, traffic above average and traffic below average.

Simulation result for traffic above average when the frequency of the requests is greater than the average shown in Table 24 and Table 25. Experimental 9 shows the improvement for average total response time by up to 6.77% and improvement for the percentage of failed requests by up to 39.15%. However, to achieve this, Experimental 9 uses the maximum number of fog devices and neglect services with the highest priority.

We also can see that the percentage of failed requests is higher for a network with 20 fog devices and 50 services. As discussed above, this happens due to increased network congestion where the requests are transmitted only between fog devices. It severely affects Experimental 2 and Experimental 6 since they use the smallest amount of fog devices; therefore, more requests go to the same devices creating more network congestions and dying by deadline compared to Experimental 8, Experimental 9. It does not produce such an effect for a network with 40 fog devices. It allows Experimental 2 to process more requests, thus, improving the average total time and percentage of failed requests while using the smallest amount of fog devices.

Table 24. Simulation results for traffic above average for scenario with 20 fog devices and 50 services.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority	Number of fog devices used
Memetic Baseline	77.8752	0.3234	119.1661	0.1475	16.8700
Experimental 2	78.7585	0.3366	130.4889	0.1466	15.8700
Experimental 6	78.7608	0.3036	133.2673	0.1584	16.1800
Experimental 8	80.2502	0.2540	131.9221	0.1607	17.2900
Experimental 9	72.5989	0.2416	144.5519	0.1836	19.0100

Table 25. Simulation results for traffic above average for scenario with 40 fog devices and 100 services.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority	Number of fog devices used
Memetic Baseline	74.7604	0.1419	109.0995	0.0874	35.3900
Experimental 2	74.5608	0.1096	97.7878	0.0619	34.5500
Experimental 6	75.4876	0.1325	111.5557	0.0946	34.7300
Experimental 8	74.7115	0.1001	97.9507	0.0517	35.8100
Experimental 9	71.8213	0.0864	105.8453	0.0887	37.3900

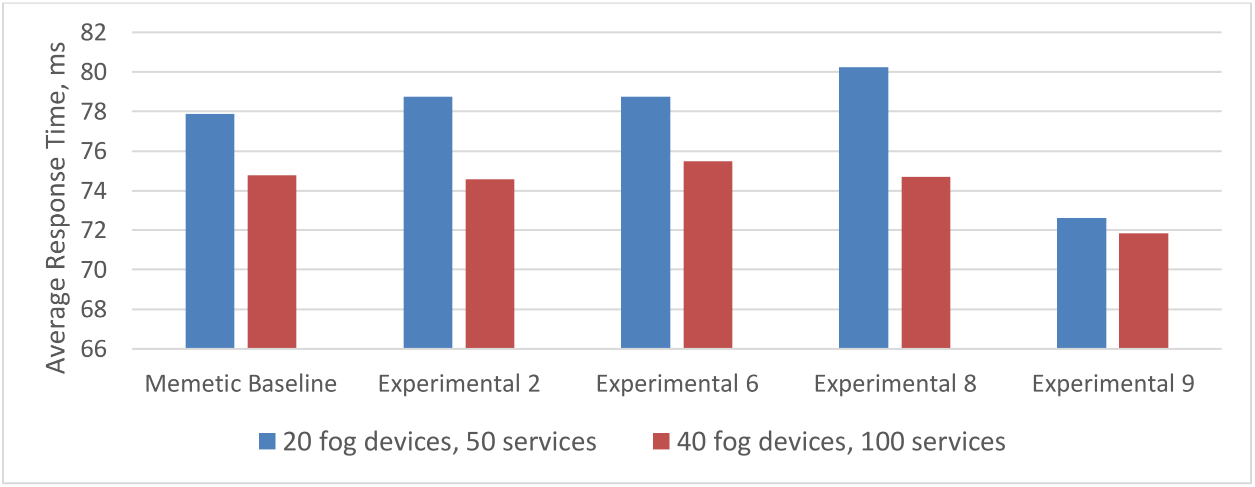

Figure 12. Average total response time for settings with traffic above average.

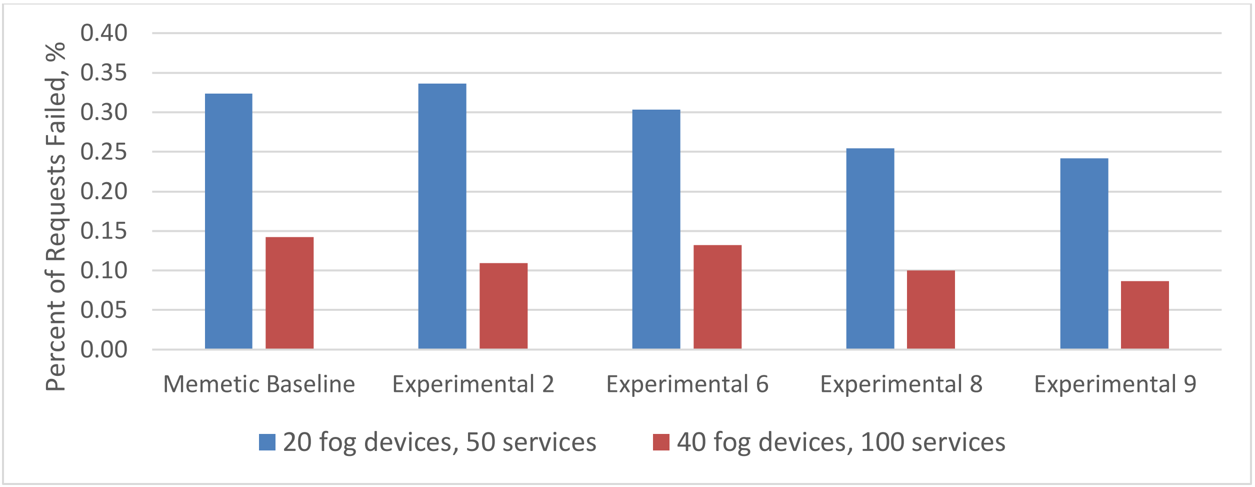

Figure 13. Average percent of failed requests for settings with traffic above average.

Simulation result for traffic below average when the frequency of the requests is smaller than the average shown in Table 26 and Table 27. The results for simulation with traffic below average show that modified memetic algorithms improve average total response time and percentage of failed requests up to 2.2% and 36.3%, respectively. Experimental 8 shows the best performance in the percentage of failed requests for all services and services with the highest priority, which can be explained by the new objective function that prioritizes services with the smallest deadlines allowing to complete more requests on time.

Table 26. Simulation results for traffic below average for scenario with 20 fog devices and 50 services.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority	Number of fog devices used
Memetic Baseline	81.0521	0.2741	139.5765	0.1528	16.8000
Experimental 2	78.6062	0.2642	134.3264	0.1377	15.8100
Experimental 6	78.5710	0.2651	144.3016	0.1805	16.0600
Experimental 8	76.8907	0.2353	126.5442	0.1431	17.0800
Experimental 9	74.2194	0.2248	182.0745	0.2522	19.1600

Table 27. Simulation results for traffic below average for scenario with 40 fog devices and 100 services.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority	Number of fog devices used
Memetic Baseline	74.4531	0.0973	99.2440	0.0562	35.3200
Experimental 2	73.9519	0.0995	102.0413	0.0782	34.4200
Experimental 6	75.5606	0.0816	84.7969	0.0364	34.5600
Experimental 8	72.8130	0.0808	89.6922	0.0442	35.7000
Experimental 9	71.2393	0.0762	107.6741	0.1045	37.0300

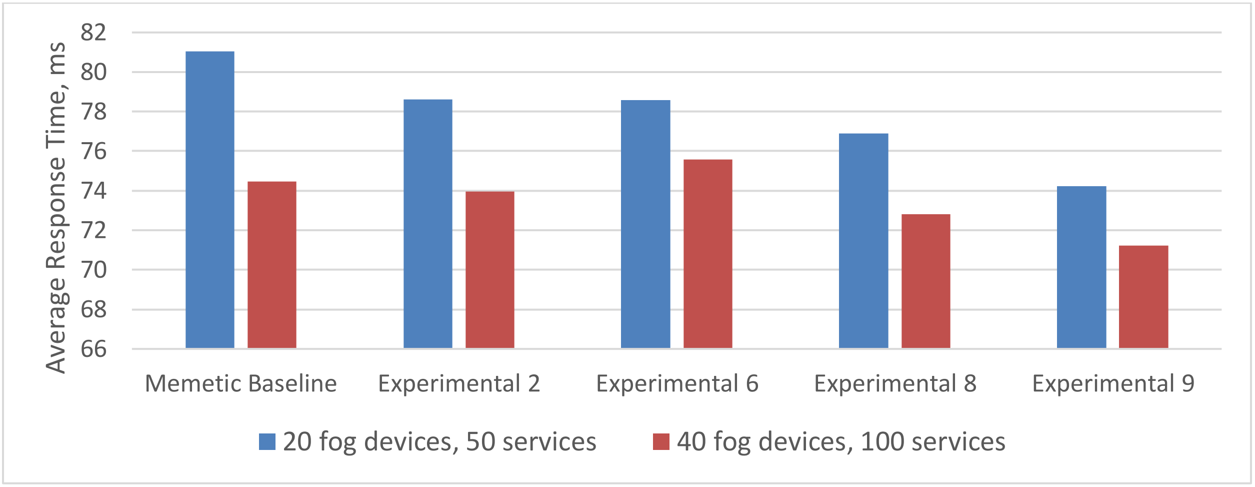

Figure 14. Average total response time for settings with traffic below average.

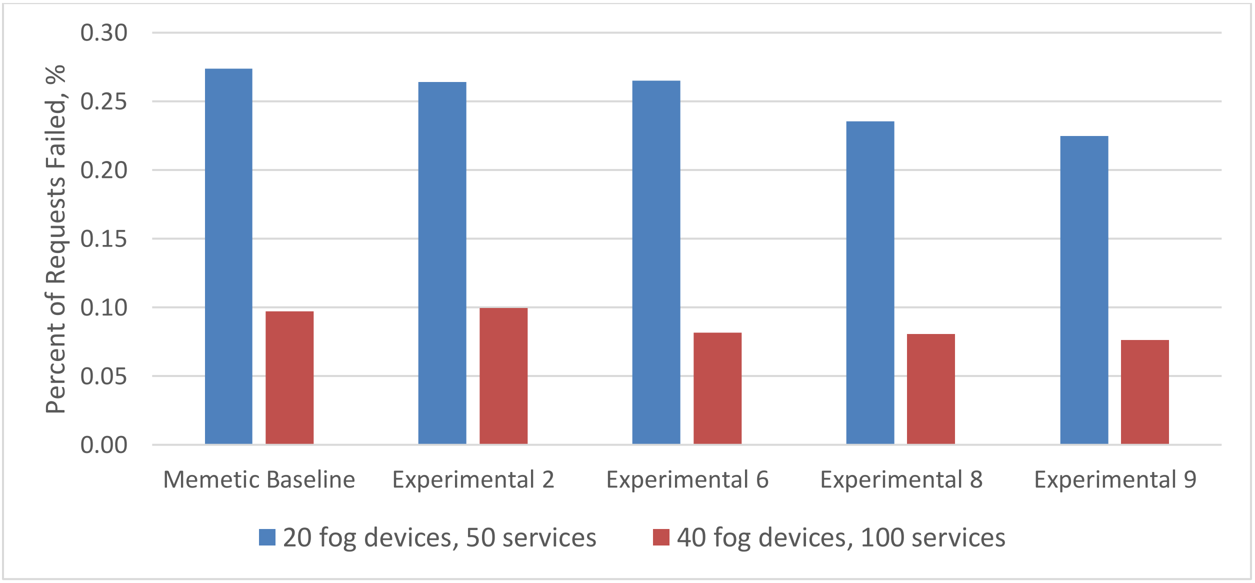

Figure 15. Average percent of failed requests for settings with traffic below average.

D. Results and Discussion for Configuration with Different Fog Devices' Resources

This subsection will discuss results for two simulation variations where we have fog devices with resources above average and fog devices with resources below average.

In the configuration with fog devices above average, fog devices have CPU, memory, IPT, and storage above average. The results of 100 simulations are shown in Table 28 and Table 29.

Table 28. Simulation results for settings with fog devices above average for scenario with 20 fog devices and 50 services.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority	Number of fog devices used
Memetic Baseline	59.7574	0.0000	60.1254	0.0000	14.8800
Experimental 2	59.7286	0.0000	60.0102	0.0000	14.8900
Experimental 6	59.9291	0.0000	60.2530	0.0000	14.9000
Experimental 8	58.9428	0.0000	59.5193	0.0000	14.9000
Experimental 9	58.3504	0.0000	58.4734	0.0000	18.2100

Table 29. Simulation results for settings with fog devices above average for scenario with 40 fog devices and 100 services.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority	Number of fog devices used
Memetic Baseline	60.1415	0.0000	60.1889	0.0000	32.3800
Experimental 2	60.1408	0.0000	60.1751	0.0000	32.5600
Experimental 6	59.8024	0.0000	59.8579	0.0000	32.2800
Experimental 8	59.2356	0.0000	59.3991	0.0000	32.4900
Experimental 9	59.0778	0.0000	59.0622	0.0000	36.1900

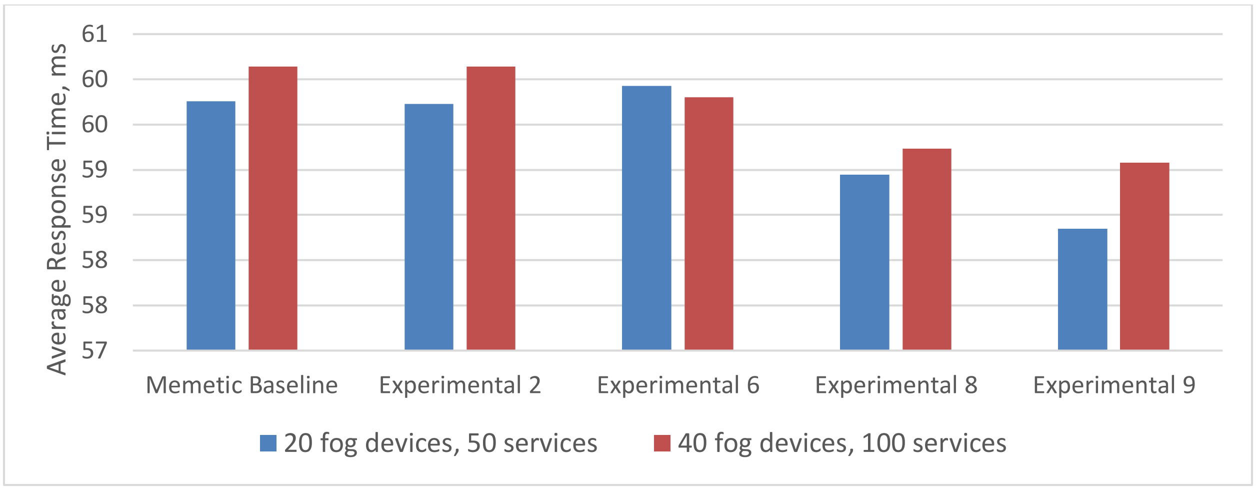

Figure 16. Average total response time for setting where fog devices' resources above average.

The results of 100 simulations for configuration with fog devices where CPI, memory, IPT, and storage are below average are shown in Table 30 and Table 31.

Table 30. Simulation results for settings with fog devices below average for scenario with 20 fog devices and 50 services.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority	Number of fog devices used
Memetic Baseline	104.5501	2.2050	992.0109	1.7518	19.2100
Experimental 2	105.3920	2.1717	966.3998	1.7028	17.0100
Experimental 6	105.8532	2.1873	1034.8908	1.8191	17.0300
Experimental 8	107.3960	2.1330	1015.4137	1.7832	19.6100
Experimental 9	106.1820	2.1536	1183.0816	2.1143	19.9300

Table 31. Simulation results for settings with fog devices below average for scenario with 40 fog devices and 100 services.

Algorithm	Average total response	Average % of failed requests	Average total response for services with high priority	Average % of failed requests for services with high priority	Number of fog devices used
Memetic Baseline	93.8274	1.1548	607.4873	0.9671	38.3500
Experimental 2	93.7423	1.1562	626.5234	1.0053	36.2400
Experimental 6	93.5901	1.1552	633.1475	1.0254	36.3400
Experimental 8	94.3539	1.1355	621.4042	0.9861	39.1100
Experimental 9	93.9570	1.1352	661.3474	1.0691	39.6100

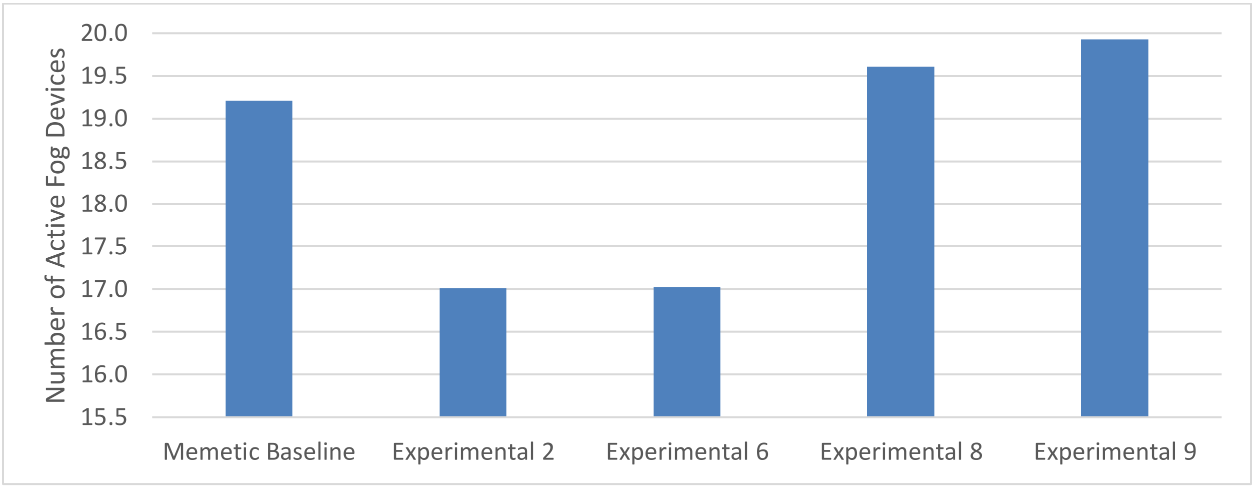

Figure 17. Average number of active fog devices for scenario with 20 fog devices and 50 services where fog devices' resources are below average.

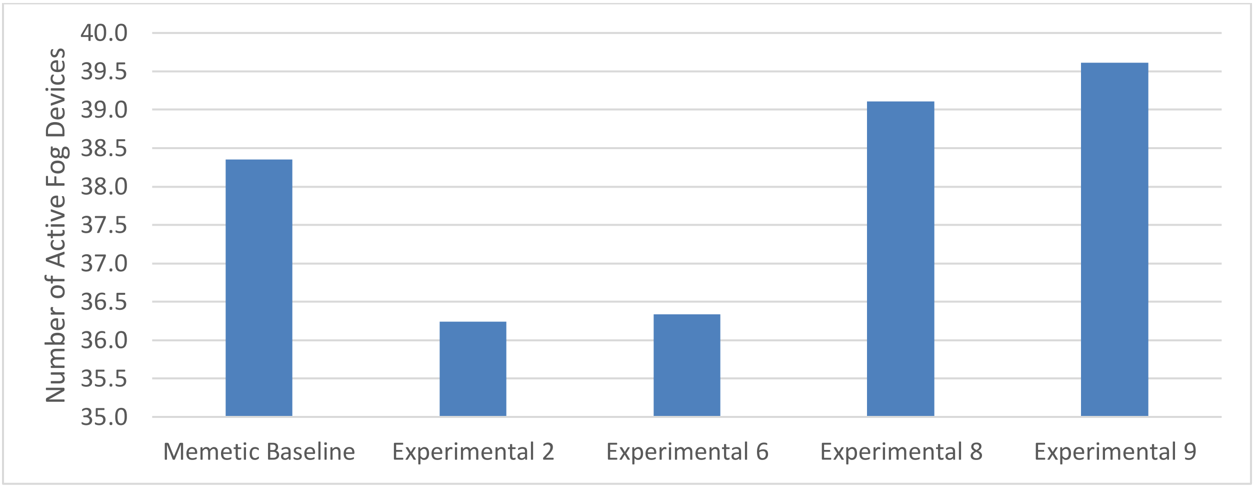

Figure 18. Average number of active fog devices for scenario with 20 fog devices and 50 services where fog devices' resources below average.

With the abundance of resources on fog devices, all algorithms perform well. Since Experimental 9 is focused mainly on average response time, it may choose the solution with more active fog devices striving to improve a single metric. In this setting, Experimental 9 improves average total response time by up to 2.35% compared to Memetic Baseline. As the fog device capacity grows, all algorithms can place all services on fog devices, making it possible to complete all the requests on time, reaching 0 in the percentage of failed requests for all services.

For settings where fog devices' resources are below average, Memetic Baseline performs the same or better than its modifications. Experimental 9 failed to improve the average total response time compared to Memetic Baseline but still found a better solution than its parent, Experimental 8. Experimental 2 performs similarly to Memetic Baseline, but choosing the same fogs nearby for users connected to the same gateway improves the number of hosts used due to the new heuristic. Thus, for settings with fog devices below average, Experimental 2 may be preferred to improve utilization and reduce the cost by allocating fewer fog devices.

E. Summary of the Best Memetic Modifications

In section we summarize the best performing modifications of Memetic algorithm and will discuss their strength and weaknesses. The comparison of best algorithms with Memetic baseline is shown in Table 32.

Table 32. Comparison of best performing modifications with Memetic baseline.

Algorithm	Average total response for all services	Average total response for services with high priority	Percentage of failed requests for all services	Percentage of failed requests for services with high priority	Number of fog devices used	Calculation time
Memetic Baseline	low	low	good	good	medium	good
Experimental 2	medium	medium	low	low	**best**	medium
Experimental 6	medium	medium	medium	medium	**best**	**best**
Experimental 8	good	good	**best**	**best**	good	low
Experimental 9	**best**	**best**	good	good	low	low

Experimental 8 is different from Memetic Baseline by having two new objective functions. We can see that it affects all metrics except calculation time. This algorithm can be useful when we need well balanced performance in all metrics but can afford to wait for placement calculation.

Experimental 9 is similar to Experimental 8. It also has two new objective function. In addition, it use machine learning model to optimize total response time as its final step. This algorithm shows the best results in total response time and percentage of failed requests for all services while providing good results for services with the highest priority. This algorithm is most applicable when small total response time and minimum number of failed requests is more important than number of fog devices used.

Experimental 6 shows near optimal performance in total response time and percentage of failed request. Its strength is the small number of fog devices used and the shortest calculation time. This algorithm is useful in case we need to calculate placement fast while keeping all the metrics satisfying.

Both Experimental 2 and Experimental 6 shows the best results in minimization of number of active fog devices. However, Experimental 6 outperforms Experimental 2. Thus, if the number of active fog devices is important, preference should be given to Experimental 6.

VIII. Conclusion and Future Work

Fog cloud computing can provide additional computational resources for IoT devices, helping save batteries and improve services quality. The optimal placement of services on fog devices is essential. It affects the response time and the number of requests filed by the deadline.

In this project, we proposed multiple modifications of the memetic algorithm to improve speed and total response time and decrease the percentage of failed by deadline requests. We consider a more realistic approach, accounting for different users' locations and request deadlines. We investigated the importance of including distance to the cloud in the objective function that minimizes the distance between user and service. The results show that the algorithm performs better when it is aware not only of the number of services sent to the centralized cloud but also how far away requests need to be sent, thus foreseeing network latency for such a placement.

The best performing algorithms are Experimental 2, Experimental 6, Experimental 8, and Experimental 9.

Experimental 2 and Experimental 6 uses a new heuristic in local search. In addition, Experimental 6 performs a local search only every second generation. These algorithms were able to use the minimum number of fog devices while keeping the optimal or near-optimal performance in other metrics. In addition, Experimental 6 improved calculation time by 30% compared to the Memetic baseline. Thus, Experimental 6 can be a good choice when we need placement as soon as possible while having satisfying performance and a minimum number of active fog devices, for example, to keep cost at a minimum.

Experimental 8 uses two new objective functions. It improves the percentage of failed requests for all services and services with the highest priority by up to 37.66% and 16.08%, respectively. It shows overall very good performance in all metrics and can be used when the calculation time is not very important, for example, if we have a long-term placement.

Experimental 9 also uses two new objective functions and machine learning optimization to find placement with the best average total response time. This approach helps to make the algorithm more flexible instead of using equal weight for all objective functions. Experimental 9 improves average total response time by 4.54%. However, it uses more fog devices and shows only near-optimal performance for services with the highest priority. This algorithm can be used when services do not have different priority levels and can allow using more fog devices to maintain a good average total response time.

Possible further research directions include finding optimal settings for a number of generations, population size, and the frequency value to how often to skip local search. Since local search is the bottleneck of the algorithm, some new heuristic or approximation can be considered to improve the speed. In our simulation, we were not able to turn off fog devices when their time was over. In our experiments, we only performed the initial allocation. The extension of this work may be enabling a migration during runtime for both services and the local orchestrator. Simulator capabilities may restrict such experiments. Thus, simulation on a real testbed may be helpful to explore the performance of architecture with volunteering devices.

Finally, for machine learning approach optimization, online learning may be introduced. It will update the ml model with results from new placement to improve the accuracy of predicted value and make the model aware of different network configurations.

This approach may be further enhanced to optimize other metrics, explore more architectures and feature engineering to make predictions more accurate.

REFERENCES

[1] "How Many IoT Devices Are There in 2021? [All You Need To Know]," *Techjury*. [Online]. Available: https://techjury.net/blog/how-many-iot-devices-are-there. [Accessed: July 31, 2021].

[2] H. Sami and A. Mourad, "Dynamic on-demand fog formation offering on-the-fly IoT service deployment," *IEEE Transactions on Network and Service Management*, 2020

[3] Z. Zhong and R. Buyya, “A Cost-Efficient Container Orchestration Strategy in Kubernetes-Based Cloud Computing Infrastructures with Heterogeneous Resources,” *ACM Transactions on Internet Technology*, 2020.

[4] C. Zhang, et al, "An Energy-aware Host Resource Management Framework for Two-tier Virtualized Cloud Data Centers," *IEEE Access*, 2020.

[5] N. Gholipour, E. Arianyan, and R. Buyya, "A novel energy-aware resource management technique using joint VM and container consolidation approach for green computing in cloud data centers." *Simulation Modelling Practice and Theory* 104, 2020.

[6] A. Khan, et al "HeporCloud: An energy and performance efficient resource orchestrator for hybrid heterogeneous cloud computing environments," *Journal of Network and Computer Applications*, 2021.

[7] K. Kaur, et al, “KEIDS: Kubernetes-Based Energy and Interference Driven Scheduler for Industrial IoT in Edge-Cloud Ecosystem”, *IEEE Internet of Things Journal*, 2020.

[8] Naha R. et al, "Deadline-based dynamic resource allocation and provisioning algorithms in fog-cloud environment," *Future Generation Computer Systems 104*, 2020.

[9] X. Ren, Z. Zhang, and S. Arefzadeh, "An energy-aware approach for resource managing in the fog-based Internet of Things using a hybrid algorithm," *International Journal of Communication Systems*, 2021.

[10] H. Wu, et al, "Collaborate edge and cloud computing with distributed deep learning for smart city internet of things," *IEEE Internet of Things Journal 7,* no. 9, 2020.

[11] Y. Chen, et al, "Deep Reinforcement Learning based Dynamic Resource Management for Mobile Edge Computing in Industrial Internet of Things," *in IEEE Transactions on Industrial Informatics*, 2020.

[12] J. Huang, C. Xiao, and W. Wu, “RLSK: A Job Scheduler for Federated Kubernetes Clusters based on Reinforcement Learning,” *2020 IEEE International Conference on Cloud Engineering*, 2020.

[13] S. Tuli, S. Ilager, K. Ramamohanarao, and R. Buyya, “Dynamic scheduling for stochastic edge-cloud computing environments using a3c learning and residual recurrent neural networks,” *IEEE Transactions on Mobile Computing*, 2020.

[14] I. Lera, C. Guerrero, and C. Juiz, “YAFS: A simulator for IoT scenarios in fog computing,” *IEEE Access* 7, 2019

[15] K. Velasquez, et al, “A rank-based mechanism for service placement in the fog,” *IFIP Networking Conference (Networking)*, 2020.