LEBANESE AMERICAN UNIVERSITY

Machine Learning Models for Scheduling On-Demand Fog Placement and Optimizing Container Deployment

By

Peter E. Farhat

[redacted]

Submitted in full fulfillment of the requirements

for the degree of Master of Science in Computer Science

School of Arts and Sciences

March 2020

ACKNOWLEDGMENT

I would like to thank my family, friends and LAU for the unlimited support given throughout this journey. Also, I would like to thank Dr. Azzam and Omar Abdel Wahab for their guidance and motivation that ensured a high quality final outcome.

I am grateful for all the opportunities and the aid provided by the university. The Lebanese American University allowed me to experience several new technologies and experiment with new ideas and concepts. Such work would have not been possible elsewhere.

I would also like to thank my committee members Dr. Haraty and Dr. Habre for their time in revising my work and providing their valuable feedback.

This work is accomplished thanks to everyone's help and support.

Machine Learning Models for Scheduling On-Demand Fog Placement and Optimizing Container Deployment

Peter E. Farhat

ABSTRACT

Fog computing is an extended cloud computing technology allowing services embedded into virtual machines or containers to be placed at the edge with closer proximity to the end devices. Nevertheless, one of the main difficulties adding up to the complexity of the on-demand fog placement topic is deciding on the proper time and place of the fog deployment process, and deriving the adequate container and service distribution over the available fogs at a certain location. Several techniques have been suggested in the current literature as potential solutions, in which some consider users preferences or random-based approach for fog deployment, while others reside on threshold-based mechanisms. However, due to the huge increase in the number of requests coming from end devices including IoT users, fog deployment and container placement must be scheduled to serve locations with high service requesting profiles while decreasing the cloud processing load. In this context, we first propose a fog placement model that allows to produce an adequate scheduling decision by using a hybrid technique combining time series forecasting and reinforcement learning. The proposed technique learns the intensity of service invocations and behavior of end devices at different locations over time for predicting the localization plan. Second, we propose a K-Means based clustering model embedded within a multi-objective optimization scheme for fog and container placement. A comparison of our proposed solution with random-based and threshold-based fog placement scheduling

approaches show that the number of processed requests performed by the cloud decreases from 100% to 29% compared to 93% and 67% in the other two models. The aforementioned results explore the efficiency of our proposed scheme in scheduling the fogs in their rightful place, which helps in decreasing the cloud's load, decreasing the network congestion, and increasing the overall quality of service. Moreover, experimental results illustrate that the proposed optimization model and clustering technique further improve the pre-existing heuristic-based solutions for container distribution.

Keywords: IoT, On-Demand Fog Computing, Cloud, Container Scheduling, Service Scheduling, Machine Learning, Reinforcement Learning, Time-Series, Clustering, Genetic Algorithm, K-means.

TABLE OF CONTENTS

List of Tables

List of Figures

Figure **Page**

Chapter 1

Introduction

In this chapter, we first introduce the motivation and the problem statement of the topic that we are working on. We then discuss the objectives that we focus on throughout this work. Finally, we explain the methodologies being used, and we state our contributions regarding both Fog Localization and Container Distribution problems.

1.1 Motivation and Problem Statement:

The emerging concept of the Internet of Things (IoT) and the rapid increase in the number of resource limited devices has triggered the creation of the fog concept by bringing services closer to the end devices [23][14][3]. With the assist of fog nodes which represent units that serve incoming requests [28], the end devices shall be able to request services with less delay, less congestion, and better quality of service (QoS) while conserving energy [7]. The main role of a fog is collecting requests from end devices, processing those requests, and sending/receiving data to/from the cloud on behalf of the end devices. This procedure will not only save the energy of those device, but will also aid the cloud by processing data locally on the fog nodes rather than processing them on the cloud. Having this said, using fogs shall result in a decrease in the network congestion and delay [37]. [25] sheds the light on the notion of volunteering devices that are used as fog nodes to

increases the coverage and availability of fogs. This approach provides flexibility regarding on demand fog creation and is enhanced using new technologies such as micro-services, containers, Docker and Kubeadm tools [19][26]. In order to create fogs on demand, a certain mechanism shall be able to detect the need for a fog at a certain location. Two main methods are currently being used in the literature. The first is the random-based approach which deploys fogs at random locations. The second is the threshold-based approach which invokes a deployment every time a certain predefined threshold is exceeded. Both of the mentioned methods are not applicable in real life due to the high number of service requests and the limited resource availability.

To illustrate the drawbacks of the above methods, suppose that we have two locations L1 requesting 1000 request at t = 0 and L2 requesting 2000 requests at t = 2. Suppose that we have the luxury of only deploying one fog. At which location and time should the fog be deployed? If we were to use the threshold-based approach with a threshold equals to 1000, the fog shall be deployed at location L1, which is not the most feasible solution. If we were to use the random-based approach, either L1 or L2 shall be picked as candidate locations, which is also not the most feasible solution. In order to come up with the most feasible localization solution, a model is needed for studying and learning the behaviour of the requests and requester for performing feasible future actions. To the best of our knowledge, there is no existing work in the literature that tackles the decision of deploying fogs using a behavioral based approach. Moreover, the on-demand fog placement field is still new and only few approaches exist in the literature [25][11]. In this regard, there is a need for improving the current models with intelligent mechanisms while addressing real life parameters and criteria.

1.2 Objectives:

This work focuses on two main objectives. The first is to schedule the creation of on-demand dynamic fogs over an unknown set of locations while considering user requests and behavior over a period of time. The advantage of that comes in the form of decreasing the cost of cloud processing, decreasing delay, decreasing network congestion, increasing QoS, and so on. The second objective is to provide an efficient way of placing containers and distributing services over the newly created fogs. This should be done in a way that maximizes a set of objectives such as increasing the quality of service, increasing survivability, increasing the number of pushed service, increasing the number of idle hosts (less cost), and decreasing the total delay between users and servers.

1.3 Methodology and Contributions:

In our previous work [11], we created a machine learning algorithm that uses purely the concept of Q-learning. The only drawback of that work, which allowed using pure Q-learning, is having a limited number of both locations and requests. On the contrary in this work, and while having unlimited locations and requests, we designed a model that is capable of studying the behaviour of requests, their location, time, and intensity to decide on the proper fog localization schedule using multiple techniques including instant updating of the Q-table values which produced better results. We designed a time decision model based on hybrid machine learning combining both time series and reinforcement learning. Time series is used to extract statistics and characteristics belonging to different timestamps [8]. Q-learning on the other hand is another area where the aim is to maximize a reward by performing correct actions [35]. In our work, we have merged both techniques into one model capable of extracting statics at different timestamps using Q-learning techniques. It is worth noting that we altered the original known concept of Q-learning to make it fit the environment that we are

working with. A comparison between the original and the derived Q-learning shall be presented in section 4.2. In this context, our model is able to predict the most feasible location and time for the fog deployment process for serving the biggest number of incoming requests. Such predictions help in decreasing the network delay and congestion, decreasing the pressure on the cloud's resources as well as increasing the QoS.

Now that we have a mechanism that ensures an adequate distribution of fogs over different locations, it is time to enhance the model suggested in [25]. This model uses a Memetic Algorithm (MA) with a set of objectives to solve the container distribution problem. A container is a standard unit of software that packages up a service and that allows the usage of the host's operating system rather than creating a virtual machine on the host for its functionality [25][11]. The Memetic Algorithm used to distribute these containers is an enhanced version of the well-known Genetic Algorithm. It executes the same procedures (mutation, crossover and selection) as of the GA to form populations, however, it adds a probabilistic search step to enhance it. Since MA is based on GA, we introduce a new version of the Genetic Algorithm which uses K-means clustering mechanism to create intelligently designed initial population, with a different set of objective functions to solve that same problem. Because of the connection between the two methods, We were able to compare both models in terms of efficiency and quality using the same set of objectives and input data. More on that in chapter 3.

This work yields two main contributions in the cloud-fog-requester environment.

- The first contribution is related to the location and time of the fog deployment process. Using a hybrid machine learning techniques which are time series and Q-learning, we were able by studying the behaviour of the incoming requests to predict the next day's schedule of the fog localization plan. Using three main parameter that are derived from the requests (the

number of requests per hour in a certain location, the total number of requests in all locations per hour and the intensity of requests which provides an image of the intensity of the incoming requests), we were able to create a model which results in accurate predictions of where and when a fog shall be created. Our model proved to be feasible and more reliable compared to other approaches suggested in the literature and was able to decrease the processing load of the cloud by deploying fogs in the most adequate locations at the most adequate times to process the incoming requests of the location.

- The second contribution of this work is the enhancement done to the container distribution model of [25]. We have introduced a new set of objectives (increase the number of pushed services, decrease the number of used hosts, increase survivabilty of hosts, decrease total delay, and increase a critical placement value), while solving this problem using GA. An enhancement is also applied on the used GA. Rather than having an initial set of parents which is composed of completely random distribution of services over hosts, we suggest a machine learning k-means clustering step to take place when setting this initial set. The clustering step serves the purpose of linking the needs of services to the possible appropriate hosts by clustering the hosts into clusters each having special characteristics that each can offer. Also, the services shall be also clustered into clusters each requesting certain needs. Rather than setting the initial parent solution to be completely random, we shall link the clusters of the services to the clusters of hosts that can serve the needs of those services. By applying such a step, we can directly and more efficiently start to generate feasible solutions that maximize the objective function compared to randomly generated set.

1.4 Thesis Organization:

This work is divided into five chapters. In chapter 2 we present some work related to both fog and service scheduling techniques. In chapter 3, we discuss the container distribution problem over a set of fog nodes using an enhanced version of the Genetic Algorithm. This chapter includes the problem definition, the methodology used, the set of objectives needed for the enhanced version of the Genetic Algorithm, and includes a set of observations related to the experiments performed. We dedicated chapter 4 to the fog localization strategy using a hybrid of Time-Series and Q-Leaning. This chapter includes a description of the suggested architecture, an explanation of the model, the used algorithms, and the equations, in addition to a set of observations related to the experimental results. The last chapter concludes this work.

Chapter 2

Background and Related Work

In this chapter, we discuss the background information needed to understand the environment that we are working in. We shall provide all the needed details and definitions which enables the reader to understand the proposed models later on in this work. Also, we present in this chapter a set of related work of today's literature concerning fog localization problem, container distribution problem, or any other work that uses proposed methods found in our work.

2.1 Background Information:

We define in this section a set of important concepts that are going to be used throughout this work.

- **Cloud Computing:** is the ability to access a pool of computing resources owned and maintained by a third party via the Internet [5]. The "Cloud" is simply a group of servers that work together to serve users. It is composed of hardware, storage, network links, and a set of services that are placed in these servers. Google Cloud is a real world example of the cloud computing concept.

- **Fog Computing:** is the paradigm extending the concept of Cloud computing to a closer proximity to the end user [30]. It is also called "Edge

Computing" where the data storage, the processing of information, and the process of servicing users are concentrated at the edge of the network instead of entirely in the cloud. Fog Computing offers various benefits such as low latency, location awareness, less network congestion, less cloud cost, and mobility support. Fogs are made up of sets of devices called fog nodes hosting services and performing processing and storage procedures.

- **Fog Localization:** is the concept of localizing fogs over locations. This process can be done statically or dynamically [11]. In the case of static localization, the locations of the fogs are predefined with predefined set of services hosted on the fog nodes. In case of dynamic placement, the fogs are distributed in an on-demand manner over a set of locations without any prior planning.

- **Time-Series:** Time series analysis applies methods for analyzing time data in order to extract meaningful statistics and other characteristics. Time series forecasting is the the ability to predict future values based on previously observations throughout time [6]. This concept is used in several fields such as quantitative finance, seismology, meteorology, geophysics, and so on.

- **Q-learning:** The goal of reinforcement learning is to learn good policies for sequential decision problems by optimizing a cumulative future reward signal [31]. The aim of Q-learning is to learn a policy, which tells an agent what action to take under what circumstances. A positive reward is given to the agent if the decision was "good", else, that agent is given a negative reward. This concept has been used in training robots and machines to perform intelligent decisions in games for example.

- **Containers:** Containers play the role of a packaging mechanism for services that can be abstracted from the environment in which they actually run on. This decoupling allows container-based services to easily run over any device while using the device's operating system [25]. Usually, containers

are compared to virtual machines in terms of performance, where the virtual machine has its own operating system. Studies show that containers are way more efficient because they apply less load on the hosting device.

- **Container Distribution:** is the process of distributing container-based services over a set of hosting devices. This process, which is NP-Hard, is used in our work while distributing containers over fog nodes, which allows satisfying a set of objectives.

- **Genetic and Memetic Algorithms:** Both Genetic and Memetic algorithms are heuristics used to solve NP-Hard problems. The Memetic algorithm is an enhanced version of the Genetic algorithm where a probabilistic search step is added to the sequence of GA procedures. GA is inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms. It is made up of populations, where generations are formed by performing procedures of selection, mutation and crossover [36]. Both GA and MA solve the problem to satisfy a group of objectives called objective functions (OF). The aim is to either maximize or minimize the final objective function of each solution in the population. The higher (or lower) the final OF, the better the solution.

- **K-means Clustering:** k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean. Each cluster is linked to one cluster-head which is the master of the cluster. Observations share common characteristics with their cluster-head if they share the same cluster [32].

2.2 Related Work:

As we mentioned in the previous chapter, deploying fogs on the fly using techniques that are random-based or threshold-based, isn't the best idea when it comes to feasible and efficient fog and service distribution. To the best of our

knowledge, no other work in the literature tackles the problem of fog and container distribution using a behavioural-based technique which resides on a hybrid machine learning concepts and clustering techniques to achieve both fog and service time and location distribution. Our architecture is based on a set of new technologies such as containers and micro-services for solving the problems. This section aims to discuss some work related to these technologies along side to other work that tackles random-based and threshold-based models for fog localization scheduling. Furthermore, this section also examines some papers that use heuristics as a way for solving service distribution problems and others that have to do with machine learning clustering techniques. First we will go through some existing surveys about the fog localization problem and its challenges and then discuss the related work for both fog and service distribution problems.

2.2.1 Fog and Service Distribution Challenges

The need for a proper way for distributing services over fogs and fogs over locations was brought up by a set of recent surveys [15][21][38]. The challenges that are pointed out are divides into two parts. The first part has to do with scheduling the proper day, time and location for deploying fogs over a set of locations. The second part is finding the best distribution of containers (i.e. the services) over the set of available fogs. In our work we addressed both problems and were able to create models that output feasible results. [25] provides a challenging model that allows services to be distributed over a set of volunteering fog nodes using a well-known heuristic called Memetic algorithm. In addition, other work in the literature [29][22] tackled the same problem using different heuristics, methods and models which also proved that heuristics are important when solving this NP-Hard problem. It is true that such work contributed in finding ways for distributing services over devices, however, non of them took into consideration the proper time and place for pushing services. In order to push services and distribute them over fog nodes, a fog must first be created. In this work we

introduce a model that handles the proper time and place of deploying fogs and uses an enhanced version of the Genetic heuristic which uses additional clustering steps to produce intelligently the initial solutions of the heuristic.

2.2.2 Service Scheduling

The authors of [40] considered the problem of task scheduling and balancing over fog devices. According to their paper, tasks shall be processed in a two layered architecture. The first layer includes embedded devices with limited resources (just like our case) that are located near the users for processing tasks. The second layer includes servers with high computational power. They claim that although these servers have the ability of processing multiple tasks that are forwarded from the embedded devices simultaneously, this process might be subject to network delay. From here, the idea of having a way of scheduling and balancing tasks on fog devices was suggested. To solve this problem, this work proposed a heuristic that aims to minimize the execution time of tasks on the fog resources. With an assumption of always having the ability of scheduling tasks to at least one embedded or server device, the results of the heuristic were promising. The only problem with this approach is that real environments usually contain a lot of tasks with limited number of devices which conflicts with the above assumption in real life scenarios. Furthermore, the suggested model only takes into consideration the current state of tasks and resources without taking into consideration the behaviour of the requesting devices nor their probable future states and actions.

The idea behind [24] is having a broker that manages the available resources that are present on both the fogs and cloud. The task scheduling technique is performed by monitoring the communication cost between the fogs and cloud. A heuristic algorithm is also proposed to output the best solution for task scheduling based on priority, communication cost and resource availability. This work, just like [40], do not take into consideration the behaviour of the requests and

only focuses on the current situation. This might not yield the best distribution solution over time. To overcome these problems and to ensure better solutions, we propose a machine learning model that learns the behaviour of requests by studying their history and predicting their future.

The appropriate load distribution over devices, the proper placement of services, and the correct decision of the number of pushed service are the three main point tackled in both [18] and [2]. The authors of the mentioned papers has formulated the three objectives using a Mixed Integer Program (MIP). Also, as in the case of the previous work, this paper did not address the behaviour of incoming requests. The decision of proper task scheduling does not lead to the most optimal solution over time.

[12] introduced a new concept which provides the ability of pushing services using the containerization technology at predefined fogs. This includes an orchestration layer that manages the running services. We have adopted this concept in our approach since containers proved to be useful when it comes to service and fogs. No other service distribution techniques were provided by this work. A work that is also adopted by out model is proposed by [16]. The authors presented a new way of having dynamic deployment of services over fog nodes using Docker. Such work gives the ability of adding, stopping and removing services off the devices at anytime. The way of pushing services in this work does not guarantee an optimal service distribution since their model never checks the behaviour nor takes into consideration the future scenarios that might take place. Without the concept of prediction and with only performing actions according to the current state, the cloud might push services with low priority at time t = 0 (current state) and end up with no space for pushing high priority service at t = 4 (future states). This will result in having degraded solutions.

[25] combines the ideas of both [12] and [16]. The author creates a model that represents a fusion of the suggested technologies and uses a heuristic called Memetic algorithm to solve the service distribution problem. In our work we reside on the work of [25] in creating a part of our model. We also try to enhance the algorithm used by using a machine learning clustering technique. This work deploys fogs on demand and wherever needed. However, they assume that the fogs are always pushed to the most needed locations. In real scenarios, some locations might request high number of requests with high importance and others may not. We need to have a way that determine the best distribution of fogs over the available locations and this is what we aim for in this work.

2.2.3 Fog Scheduling

The action of creating fogs after receiving a request stating the need of the deployment at a certain location is considered to be a random-based approach for fog creation. The authors in [17] proposed a fog placement method which depends on the user to request the fog placement. This approach is not feasible in real environments since we have a limited number of fogs while giving access to infinite number of devices to request the fog creation mechanism. Because of the randomness of the human behaviour, this approach falls under the umbrella of random-based approaches.

If a fog is overloaded with processing tasks, an alternative shall be given to the devices that are requesting from that fog. [39] tackled this problem by routing the requests to other neighboring fogs in case the requested fog is unable to process further request. The authors of this paper proposed a mechanism which initially calculates the estimated waiting time of the incoming requests then decides on whether the requests should be processed by the current fog or get routed to another one. Setting a delay threshold is equivalent to setting a threshold on the number of incoming requests. From our point of view, the threshold-based

approach is decided using the number of incoming requests to the cloud. If the number of request is more than a defined threshold then the cloud requests a fog to be deployed in the source location of the requests.

2.2.4 Time Series and Q-learning

The authors of [13] suggested a hybrid method that joins both time series and reinforcement learning to predict the best solution for a trading game. They introduced a Q-table having static number of actions and dynamic number of states. Unlike the mentioned paper, the hybrid method that we use aims to solve a different problem. Also, the Q-table used in our approach consists of static number of states (the hours of the day) while having dynamic number of actions (creation of fogs at a specified location). Given our format, the machine shall be able to learn the behaviour of requests and shall predict the most appropriate location and time to deploy fogs.

In our previous work [11], we proposed a pure reinforcement learning strategy to predict the proper time and location for the fog deployment process. We were able to decrease the number of processed request by the cloud from 100% to 30% and we showed that the proposed strategy excelled in the domain compared to other suggested approaches such as random-based and threshold-based. In the current work we aim to further decrease this number by using another technique which is able to predict and periodically update the Q-table to cope with the unknown number of locations that might request services and with the sudden changes in the environment (An event that suddenly took place in a certain location for example).

2.2.5 Machine Learning - Clustering

In [33] the authors proposed a solution for solving the placement and readjustment process of Virtual Network Functions (VNF). According to this paper, VNFs

helps in improving the management of the network by switching to software-defined components rather than having expensive dedicated hardware. Avoiding the burden of using hardware appliances which require specialized managing personal,a lot of energy to function and are considered to be short-lived, helps ease the life of the service providers. To go back to the main relation between our work and this paper's methods, this work proposed a machine learning approach that slices the network into a smaller set of disjoint clusters. According to them, this approach helps in reducing both setup latency and complexity of VNF placement and readjustment. Three main steps are executed to achive the final placement and readjustment results. The first is applying a K-medoid machine learning clustering technique which clusters the network of devices into a set of clusters according to a set of characteristics such as CPU, Bandwidth, Energy and Delay. The second is selecting the bests cluster heads which reduces the clustering time and improves the quality. The third step is using another machine learning model for the placement and readjustment of the VNFs. As we can see, the clustering technique was only used to improve the efficiency of having a final result by giving the ability of applying ILP formulation which decreased the complexity of this NP-Hard problem. We can reduce the problem solved by this paper into our own problem. This work tends to place functions over a set of devices on the cloud which can be reduced to our problem which aims to place a set of services on fog devices. Rather than just clustering the devices found on the fog according to a set of characteristics, we aim to do the same for the services. Once this is done, we will have a set of clustered devices that can offer some characteristics (such as high CPU) while having on the other hand a set of clustered services that require some characteristics (such as the need for high CPU). Once we have these cluster sets, we can then directly link the needs to the offered characteristics which saves us from randomly choosing initial parents in our enhanced algorithm and which helps in improving the algorithm.

Chapter 3

Cluster–Based On Demand Container Placement using Machine Learning

In this chapter we complete the overall vision of having on-demand fog creation with adequate service distribution over fog nodes. The service distribution problem is not a new topic in the literature. Many papers have contributed in their own way to enhance solving this NP-Hard problem. This work suggests the usage of machine learning to enhance one of the famous techniques that are used to solve such a problem. We tend to use K-means to cluster and enhance the initial population of the well-known Genetic Algorithm, which proved to boost the classical GA and outperform other algorithms, such as Memetic algorithm, in terms of convergence efficiency and solution quality.

3.1 Problem Formulation and Methodology

As described in figure 1, the mid layer (containing the fog) is made up of a set of fog nodes that are in our case volunteering devices such as laptops, desktops, phones etc. The final missing piece that completes our puzzle focuses on the distribution of containers that are provided by the cloud on a given set of vol-

unteering devices. In [25], the author solved this issue by launching a Memetic algorithm heuristic which performs a set of selection, crossover, and mutation operations on a randomly generated population of solutions, resulting in a solution that maximizes the objective of that heuristic. What we plan to do in this work is to use the Genetic Algorithm (the parent of the Memetic Algorithm), with a clustered initial population to solve the problem. We believe that creating a clustered initial solution shall force the algorithm to focus on the solutions that are most likely to be the best candidates of this problem and thus gain more time and accuracy. We believe that we shall be able to converge faster and that the quality of the final solution shall be better compared to the Memetic Algorithm, where the initial population is made up of randomly generated solutions. In the next sections we shall define the container placement problem and the methodologies used in the Genetic Algorithm to solve it.

3.1.1 Problem Formulation:

Given a set of services $S = \{S_1, S_2, ..., S_n\}$ that are represented using containers, our goal is to find the best way to distribute those containers over a set of volunteering hosts $H = \{H_1, H_2, ..., H_n\}$ which are represented by the fog nodes.
This problem is NP-Hard and cannot be solved in linear time. To prove this claim, let's map our problem to the Bin-Packing problem which is a known NP-Hard problem according to [10]. Bin-packing can be described as having a set of different sized objects that needs to be packed into a set of bins. The objective here is to fill the maximum number of items while using the minimum possible number of bins. We can easily notice the link between this and the container placement problem where two of its objectives are maximizing the number of containers pushed onto the fog nodes (which represents the number of items in the Bin-packing problem) and minimizing the number of used volunteering nodes (which represents the bins in that problem). Since we are dealing with an NP-Hard problem, we can go ahead and use the proposed Genetic Algorithm to solve

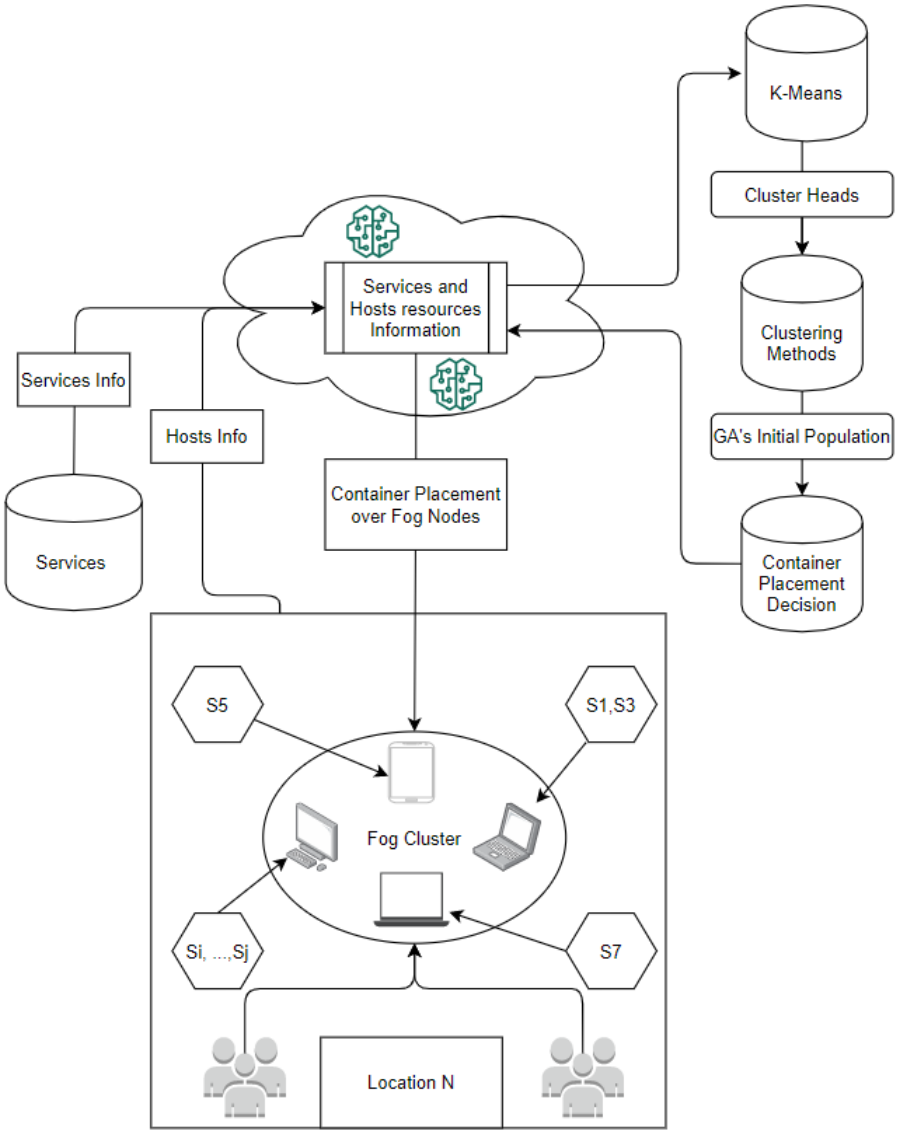


Figure 1: Container Distribution Model

it.

In order to provide a solution that takes into consideration the resources that are being used and the quality of the generated service, we have decided to include five different objectives, where the Genetic Algorithm tries to maximize their values while producing high quality results.

Theoretically, every device that wishes to play the role of a volunteer in the fog environment must share some basic information about its resources with the cloud. Similarly, the information of every service must be known by the cloud prior to algorithm execution. Below we introduce the definition of the needed host and service information:

1. H_i = [*H-cpu, H-energy, H-memory, H-delay*] $\forall i \in 1, ..., |H|$ ***Where:***

   H-cpu: is the total CPU power that can be distributed over the pushed containers on fog device H_i.

   H-energy: represents the Energy of device H_i. If this value reaches 0 then the devices will no longer be active.

   H-memory: is the total Memory space that can be distributed over the pushed containers on H_i.

   H-delay: is the delay that can be provided by host H_i to each user.

   |H|: is the number of hosts.

2. S_j = [*S-cpu, S-energy, S-memory, S-delay, S-priority*] $\forall j \in 1, ..., |S|$ ***Where:***

   S-cpu: is the CPU power needed by service S_j to function normally.

   S-energy: represents the amount of Energy consumed by S_j while executing services on H_i over time.

   S-memory: is the Memory space needed by S_j to function normallyH_i.

   S-delay: is the delay required by S_j to have the best quality experience.

   S-priority: is a value between 1 (low) and 5 (high) that represents the importance of the presence of S_j in the fog near the users.

   |S|: is the number of services.

When all the host and service information is provided to the cloud, the calculation of five different objective function shall be made possible. The best solution Kb of the solution set K_v $\forall$ v $\in 1, ..., |K|$, where |K| is the number of solutions in the current generation, shall contain the highest objective function **OF** in terms of all the defined metrics of that generation. **OF** is calculated by adding up the five sub-objective functions $\{F_1,...,F_5\}$ while normalizing their values and multiplying each by the weight assigned to it. Below we present the main objective function **OF** alongside with the five sub-functions that constitute it.

Objective Function:

$$OF\ (K_v) = [w1{*}F_1(K_v)] + [w2{*}F_2(K_v)] + [w3{*}F_3(K_v)] + [w4{*}F_4(K_v)] + [w5{*}F_5(K_v)] \tag{3.1}$$

Where:

- $w \in [1, ..., 5]$: represents the weight for each sub-objective function respectively where w1 + w2 + w3 + w4 + w5 = 1.
- F_1: calculates the number of pushed services/containers on the fog nodes.
- F_2: calculates the number of hosts hosting no services (idle hosts).
- F_3: calculates the number of active hosts over time (survivability).
- F_4: calculates the impact of service placement (critical placement).
- F_5: calculates the total delay of the environment after service distribution.

1. F_1- *Number of Launched Services Calculation:*

$$F1(K_v) = max((\sum_{i=1}^{|H|}\sum_{j=1}^{|S|} K_{ij}) \times w1) \tag{3.2}$$

Where:

- K_{ij}: Is 1 if a service exists on H_i, 0 otherwise.

The aim of this function is to calculate the total number of services being pushed to the fog nodes. Having a solution K_v that maximizes the number of pushed services will guarantee a high value of F_1. As we maximize the number of pushed services, the value of F_1 will increase since it will then tend to give access to a wider range of users that demand using different kinds of services. This function takes as an input the solution produced by the Genetic Algorithm that gives the maximum number of services and outputs a value normalized over 1, which represents the goodness of the solution with regards to the number of pushed services. Algorithm 1 is a pseudo code that illustrates this procedure. In this algorithm we loop over the volunteering fog nodes. For each host, if it contains a service then we shall consider this service as visited and increment the count. As a side note, here, we assume a service can only be pushed once to the same fog. For example, if we already pushed service S_4 to H_1 then we cannot push service S_4 to H_1 again (or to any other host H_i).

Algorithm 1 F_1: Number of Launched Services Calculation

```
Inputs: K_v, MaximumNumberOfServices
Output: Objective Function value for F_1
Procedure Number of Launched Services
SeriveCount = 0
for Each Host as H_i do
    if H_i hosts any service S_j then
        Mark S_j as launched
        SeriveCount = SeriveCount + 1

OF = SeriveCount/MaximumNumberOfServices
```

2. F_2- *Number of Idle Hosts Calculation:*

$$F2() = max((\sum_{i=1}^{|H|} Hidle_i) \times w2) \tag{3.3}$$

Where:

- $Hidle_i$: Is 1 if the host is idle, 0 otherwise.

The aim of this function is to calculate the number of hosts that contain zero services. As the number of idle hosts increases, the cost of fog creation decreases. Increasing the value of this objective function will give us the ability of serving users with less hosts and consequently less cost. The calculation of this objective function is described in algorithm 2. This functions takes as an input the solution K_v with the maximum known number of hosts. While we loop over the hosts, if we have a host that contains zero services then we increment our counter. At the end, we output the value of that counter divided by the total number of hosts. As the counter value increases towards maximum, the objective functions increase towards value 1, which is the highest value a sub-objective function can reach.

Algorithm 2 F_2: Number of Idle hosts Calculation

1: **Inputs:** K_v, *MaximumNumberOfHosts*
2: **Output:** Objective Function value for F_2
3: **Procedure** Counting Idle Hosts
4: *HostCount = 0*
5: **for** `Each Host as` H_i **do**
6: **if** H_i hosts no service S_j **then**
7: *HostCount = HostCount + 1*
8:
9: *OF = HostCount/MaximumNumberOfHosts*

3. F_3- *Survivability Calculation:*

$$F3() = max((\sum_{i=1}^{|H|} Hsurvived_i) \times w3) \tag{3.4}$$

Where:

- $Hsurvived_i$: Is 1 if the host did not consume all its energy, 0 otherwise.

The aim of this function is to test the survivability factor of each host. As the number of survived hosts remains constant, the fog will continue serving users with the same intensity and efficiency. From this point of view, we need to ensure that the hosts stay alive throughout the serving process. Survivability is directly linked to the energy of the host. If a host loses all of its energy while serving, then it will die. In this work, we consider that the energy needed by the service is obtained using historical information. This means that if S_j consumed x amount of energy in the past then S_j, over all time, needs at most x amount of energy to process all requests coming from all users. Having said that and as described in algorithm 3, we tend to count the number of hosts that have an energy value grater than zero after hosting the services that are defined by solution K_v. We return the number of survived hosts divided by the total number of hosts which increases towards 1, as the number of survived hosts increases towards the maximum number of hosts.

4. F_4- *Critical Placement Calculation:*

$$F4(K_v) = max((\sum_{j=1}^{|S|}(\sum_{i=1}^{|H|}\sum_{u=1}^{|U|} NoR_{ju} \times K_{ij}) \times P_j)) \times w4) \tag{3.5}$$

Where:

Algorithm 3 F_3: Survivability calculation

Input: K_v, *MaximumNumberOfHosts*
Output: Objective Function value for F_3
Procedure Counting survived Hosts
HostCount = 0
for `Each Host as` H_i **do**
    if H_i survives after hosting a set of services S_j **then**
        HostCount = HostCount + 1

OF = HostCount/MaximumNumberOfHosts

- K_{ij}: Is 1 if a service exists on H_i, 0 otherwise.
- NoR: Is the number of requests done by u to S_j.
- P_j: Is the priority of S_j.

The aim of this function is to calculate the critical placement value which represents the importance of pushing high priority services and services that are the most requested by users. Critical placement contains two main factors. The first is the high priority factor. As mentioned previously, each service contains a priority value which ranks the importance of its presence near users. Examples of such services might be ones that require very low delay to produce high quality results. The second factor is the number of requests done by users on that service. We consider that if the number of requests on service S_j is high then this service should be next to the requester for better quality of service. In general, algorithm 4 aims to increase the quality of service (QoS). In this algorithm we loop over all services and all hosts. If the service is pushed on a host, then we increment the critical placement value by adding to it the multiplication of the number of requests by the priority of that pushed service. This value will increase as the priority increases and as the number of requests increases. This increases towards 1 as the critical placement value increases towards the highest critical value scored by any previous solution in any of the generations.

Algorithm 4 F_4: Critical placement calculation

Input: K_v, *MaximumCriticalPlacementValue*
Output: Objective Function value for F_4
Procedure Calculating critical placement value
CPValue = 0, NumberOfRequests = 0
for *Each Service as* S_j **do**
    for *Each Host as* H_i **do**
        if S_j is pushed on H_i **then**
            NumberOfRequests += NumberOfRequestsToSi
    *CPValue += NumberOfRequests * PriorityOfSi*
    NumberOfRequests = 0

OF = CPValue/MaximumCriticalPlacementValue

5. F_5- *Total Delay Calculation:*

$$F5(K_v) = max((\sum_{j=1}^{|S|}\sum_{i=1}^{|H|}\sum_{u=1}^{|U|}(D_{Si} - D_{iu})) \times w5) \tag{3.6}$$

Where:

- K_{ij}: Is 1 if a service exists on H_i, 0 otherwise.
- D_{Si}: Is the delay required by the service.
- D_{iu}: Is the delay between the host and the user.

The aim of this function is to calculate the total delay after the service distribution. Each service requires some delay to produce high service quality values. When we place service S_j that requires a delay of 60 ms over host H_i, and the delay between H_i and user u is 50 ms, then if u requests service S_j, we will end up in having a delay again of 10 ms (60-50=10). However, if the delay between the host H_i and user u was 90 then the user will have higher delay of -30 ms (60-90=-30). The aim here is to increase the total delay value to ensure that the services are placed on hosts that minimize delay. We illustrate this strategy in algorithm 5 where the total delay is calculated by adding to it the subtraction of the delay requested by the

service and the delay between the requester and the host of that service. As the total delay value increases towards the best achieved total delay by any other solution, the objective function increases towards 1.

Algorithm 5 F_5: Total delay calculation

```
Input: K_v, MaximumTotalDelayValue
Output: Objective Function value for F_5
Procedure Calculating total delay value
TDValue = 0, DelayPerSi = 0
for Each Service as S_j do
    for Each Host as H_i do
        if S_j is requested by user u and S_j is placed on H_i then
            DelayPerSi += DelayOfSi - DelayOfHiu
    TDValue += DelayPerSi
    DelayPerSi = 0

OF = TDValue/MaximumTotalDelayValue
```

Algorithm 6 OF: Final OF Calculation

```
Input: Solution K_v, w1, w2, w3, w4, w5
Output: Final OF of K_v
Procedure Objective function calculation
Get output from algorithms 1,2,3,4 and 5 and save them in v1, v2, v3, v4 and v5
Final OF = (w1*v1)+(w2*v2)+(w3*v3)+(w4*v4)+(w5*v5)
```

Hard Constraints:

In this section, we present the different constraints that makes a solution feasible.

1. *CPU Constraint:*

$$(\sum_{i=1}^{|H|}(\sum_{j=1}^{|S|} K_{ij} \times S_{jc}pu) \leq H_{ic}pu) \tag{3.7}$$

Where:

- K_{ij}: Is 1 if service S_j exists on H_i, 0 otherwise.
- $S_{jc}pu$: Is the CPU power allocated to service S_j.
- $H_{ic}pu$: Is the total CPU power provided by host H_i.

2. *Memory Constraint:*

$$(\sum_{i=1}^{|H|}(\sum_{j=1}^{|S|} K_{ij} \times S_{jm}emory) \leq H_{im}emory) \tag{3.8}$$

Where:

- K_{ij}: Is 1 if service S_j exists on H_i, 0 otherwise.
- $S_{jm}emory$: Is the Memory allocated to service S_j.
- $H_{im}emory$: Is the total Memory provided by host H_i.

Now that we have presented five sub-objective functions that pinpoints solutions having high QoS, high surivability factor, low delay, low cost and high service coverage, we can move on and select the best solution by adding those five sub-objective function where each is multiplied by it's weight. The higher the value the better the solution.

3.1.2 Proposed Approach Using Clustered Initial GA Population:

We believe that the Genetic Algorithm (the root of MA), can be enhanced in such a way that orients the focus of the algorithm to produce better results in a more efficient and faster pace compared to the MA. The initial random population that is used to start the search for the best solution of the problem makes the area of search very wide. Therefore, our main research question here is: Is there any way a GA search with a sculpted initial solution can have a good quality initial population rather than randomizing? The answer to that question is yes. In this work we introduce the concept of machine learning via clustering both hosts and services using their given characterises which allows us to link the needs of services to the capabilities of the hosts. The clustering technique and its integration with the Genetic Algorithm shall be covered in the next section.

3.2 Clustering

Clustering is the process of grouping together items that share some common characteristics. It has been used for several problems such as document classification, crime identity localization, insurance fraud detection and other problems [20]. There exist several types of clustering techniques such as Agglomerative Hierarchical Clustering, Expectation–Maximization (EM) Clustering, Mean-Shift Clustering, K-Means Clustering and more where each excels under some circumstances. Our aim is to distribute containers in the best possible way over the hosts. Clustering can offer a great way of creating an initial population for the GA by linking services that for example require low delay to the hosts that can offer low delay to users. How will this help the GA in creating better solutions? Well, if we have a mechanism that allows us to determine the needs of services (by clustering them into high, medium and low clusters), then we can easily link those needs to the appropriate hosts that can provide for them. The hosts are also clustered into high, medium and low clusters in terms of characteristics. This linking process allows all the unnecessary linkages between services and hosts to be excluded which enhances the algorithm.

Services that require high cpu power shall directly be linked to the hosts that can offer such cpu power. Services that require high memory space shall be linked directly to the hosts that can offer high memory space and so on. This, and unlike the random initial population that is used by the classical GA, shall produce an intelligently designed initial population that associates demands to appropriate resources.

3.2.1 Clustering Strategy:

Our clustering method is divided into two main parts which are hosts and services clustering. For each part we intend to cluster its resources into three categories which are high, medium and low. Each category is divided into sixteen parts. In this work we have four resources that we care about which are cpu, energy,

memory and delay. Figure 2 demonstrates an example of the distribution of the sub-clusters within each category. This distribution is applied on both the hosts and services. We have decided to use the same strategy as of K-means [4] to find a set of nodes known by the name "cluster heads". In our model, we mention two different notions which are cluster heads and anchors. Cluster heads are the output of the K-means process. To obtain cluster heads for each of the four resources, we have to run the K-means on each resource, which yields twelve cluster heads. Anchors on the other hand are virtual hosts representing characteristics of the H-M-L clusters. We extract the characteristics of the twelve cluster heads (high, medium and low), and we combine them into three anchors which allows future comparison with actual hosts 3. The process of searching for the cluster heads is machine learning based. A cluster head is the master of a cluster, where all of the objects linked to it share the same characteristics. When we launch the K-means algorithm with an input of K = 3 cluster heads, it will search for 3 different nodes that can represent three different clusters. In our case, this strategy shall generate cluster heads representing the three categories which are high, medium, and low. It is used to find cluster heads that can represent high, medium, and low clusters for each of the resources (cpu, memory, energy, and delay). The end result of the algorithm is 12 cluster heads, each containing the most adequate value to represent clusters of different categories and resources. Algorithm 7 demonstrates the process of choosing the cluster heads for a specified resource. Once we obtain the needed set of cluster heads, we can go ahead and create virtual hosts and services as anchors where a comparison is done between the resources of the anchors and of that of services and hosts. This shall allow the mapping of hosts and services to the closest anchors regarding resource characteristics. Suppose that host *A* has high cpu and that we have three anchors *X, Y, and Z* which have high, medium and low cpu respectively. When we perform our clustering strategy, we measure the distance between the cpu of host A with respect to the three anchors X, Y and Z. Since A has high

cpu power, the distance between A and X shall be less than the distance between A and the other two. This means that host A shall be linked to the cluster of X with respect to the resource which is *cpu*.

To elaborate more on this point, we shall provide a detailed example which shows how the process of clustering is practically done. As mentioned above, the two parties (hosts and services) are divided into three categories (high, medium, and low (H-M-L)), which are divided into sixteen sub-clusters. Figure 3 illustrates an example of the process of mapping hosts (or services) to the appropriate anchors (or cluster heads). Cluster-heads are generated using K-means and their results are stored in anchors. When the mapping process starts, the distance between the object (host or service) and the anchor is measured. If the cpu belongs to the anchor of the high cpu cluster, this means that in the high category we will give this host a value 1 in the encoding of the host. An encoding is simply a four digit binary number and it is used as an identifier of the sub-cluster of each host or service that it belongs to. We have three encodings for the three categories (H-M-L). This is shown in figure 3. We have already mentioned that we have sixteen sub-clusters. The four digits of the binary encoding are sufficient to represent the host in which sub-cluster since 4 to the power 2 = 16 (an array with 16 cells where the index of each cell is the decimal value of that binary encoding. Ex: M-0100 (CEMD respectively) is the binary encoding of having medium energy only which will be placed in cell 2). Below is an example of the final clustering arrays of six different hosts given IDs from 0 to 5:

$$\begin{bmatrix} & \emptyset & C & \dots & EM & \dots & CEMD \\ High & host(s) - 2,3,5 & - & \dots & - & \dots & host(s) - 0,1 \\ Medium & host(s) - 0,1,3 & host(s) - 5 & \dots & host(s) - 2,4 & \dots & - \\ Low & host(s) - 0,1,4 & host(s) - 2 & \dots & host(s) - 5 & \dots & host(s) - 3 \end{bmatrix}$$

The above array states that both hosts 0 and 1 have high cpu, energy, memory and delay. This means that they should only belong to the empty arrays of the

Algorithm 7 k-means

Input: *K, SetOfNodes, Resource*
Output: Cluster heads w.r.t the Resource
Procedure Cluster Head Search
Pick randomly K nodes
while Means $M_1, M_2, ..., M_k$ are changing **do**
    for `Each node in SetOfNodes` **do**
        Calculate distances from node to each cluster heads w.r.t Resource: $(Node_resouce - ClusterHead_resource)^2$
        Add each node to the closest cluster head
    Calculate the mean $(M'_1, M'_2, ..., M'_k)$ *of each cluster:* $\frac{(Node_resouce_1 + ... + Node_resouce_n)}{NumberOfNodes}$
    $M_1, M_2, ..., M_k = M'_1, M'_2, ..., M'_k$ *respectively*

medium and low categories (i.e. encoding = 0000). Host 3 on the other hand belongs to the cluster of low cpu, energy, memory and delay. This means that it shall belong to the empty sets of the medium and high categories. Host 2 belongs to empty set of high category. This means that it has either medium or low characteristics. As we can observe, host 2 belongs to the cluster which has low cpu and to the cluster that has medium energy and memory. Logically, because of the given information in this matrix, we can deduce that host 2 also belong to the low delay cluster which is not shown in the example, and so on.

3.2.2 Clustered Initial Population Creation:

Now that both the hosts and services are distributed in their appropriate clusters, we can go ahead and apply a set of methods to place services onto the appropriate hosts. This allows services to be pushed on the hosts that can provide the appropriate power rather than distributing them over all devises. The reason behind creating the below set of methods is to give variation between solutions that belong to the same population, where each generated solution focuses on one or more objectives. In table 1, we categories the methods with respect to the objective functions of the GA. For example, method 1 guarantees that three of the objective functions (pushed services, idle hosts, and survivability) shall be maximized while forming the solutions. The intention behind this variation is

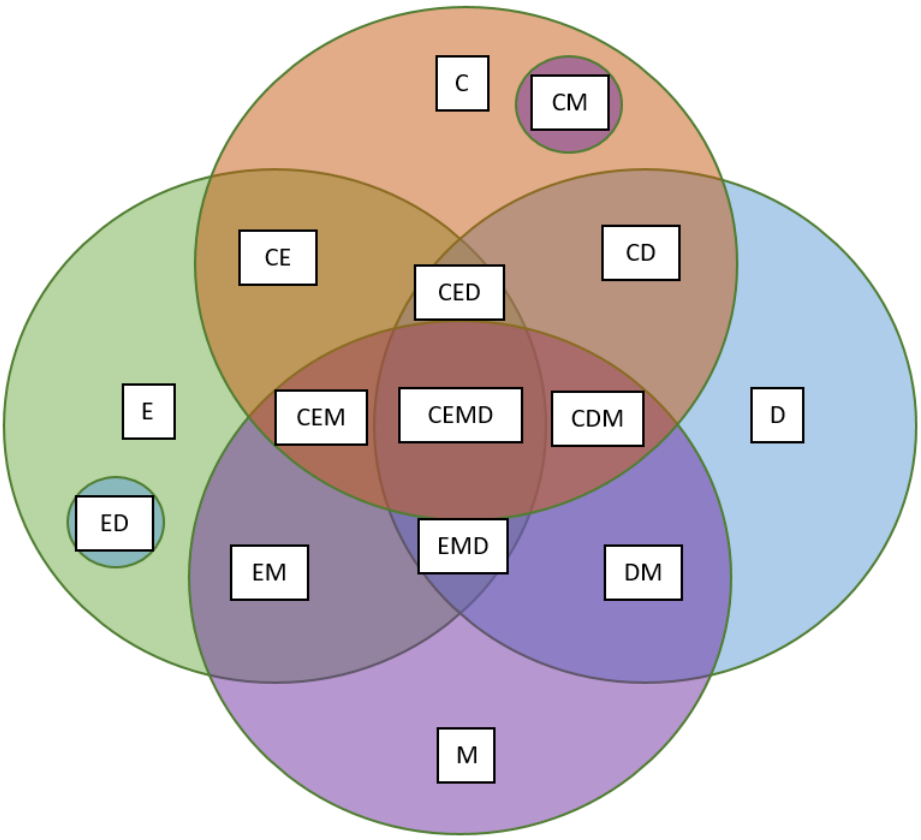
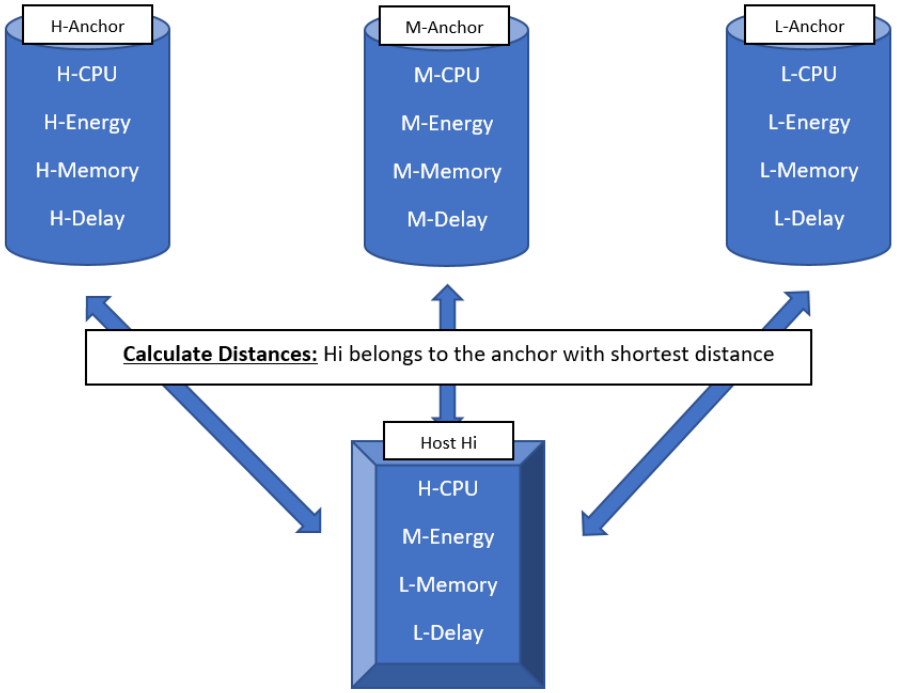


Figure 2: Cluster distribution where C = cpu, D = delay, M = memory, and E = energy



Encoding for H-M-L clusters						
Category	CPU	Energy	Memory	Delay	Binary	Decimal
H	1	0	0	0	1000	1
M	0	1	0	0	0100	2
L	0	0	1	1	0011	12

Figure 3: Anchor mapping with encoding creation

to give the GA a better chance of generating better offspring by applying the selection, mutation, crossover, and biPolarSwap operations over solutions that guarantee different objectives.

1. *Random Service Selection with High to Low Distribution over Hosts:*
   This function sorts the hosts in the order of power (according to the H category cluster, the more 1s in the encoding, the stronger the host is), randomly picks a service from the service set, and then places that service on the most powerful and free host.

   - Sort hosts from high to low according to the H-cluster
   - Randomly select a services
   - Map the services starting with the hosts containing the highest power ((in the encoding, the more the 1s the more powerful is the host) Ex: hosts that belong to sub-cluster 1111 are the most powerful hosts because they can fulfill all requirements in a more powerful way than the others)
   - If the host is full (exceeded energy, memory or cpu) then start filling the hosts next highest power
   - Repeat the same procedure until all services are visited

2. *High to Low Service Selection with High to Low Distribution Over Hosts:*
   This function is very similar to the one before the only difference is that rather than randomly selecting the services, we tend to sort them from highest to lowest (according to the H category cluster). This will allow services that require high power to be pushed first to the hosts that can provide high power (cpu, energy, memory and delay)

   - Sort hosts from highest to lowest according to the H-cluster
   - Sort services from highest to lowest according to the H-cluster

- Map the services that require the most power starting to the hosts that can provide the highest power
- If the host is full (exceeded energy, memory or cpu) then start mapping the hosts with the next highest power
- Keep on doing the same procedure until all services are visited

3. *Critical Placement Service Selection with High to Low Distribution Over Hosts:*

   This function is also quite similar to the ones preceding it i.e., the services are sorted according to the critical placement value. This will allow the services that can provide more critical placement value to be pushed first.

   - Sort hosts from high to low according to the H-cluster
   - Sort services according to the critical placement value
   - Map the services with highest critical placement value starting to the hosts that can provide the highest power
   - If the host is full (exceeded energy, memory or cpu) then start mapping the hosts with the next highest power
   - Repeat the same procedure until all services are visited

4. *H-M-L Encoding Vote:*

   Pick services randomly and let the H-M-L clusters vote on where the services must be placed. This method allows services to be directly linked to the hosts that can provide their needs (unlike the above methods where we are pushing always from high to low).

   - Randomly pick a service
   - For each of the host categories (H-M-L), compare the encoding of the service categories with that of the host categories (Ex: ServiceEncoding-H = 1010, ServiceEncoding-M = 0001, ServiceEncoding-H = 0100 with

HostEncoding-H = 1000, HostEncoding-M = 0110, and HostEncoding-L = 0001). When comparing the same categories with one another, if the numbers of the same index match then we give +1 to that match (Example ServiceEncoding-H = 1010 and HostEncoding-H = 1000, then the final grade is 3/4 since the first, second, and fourth indices match while the third is different. This implies that 3 out of 4 similarities are detected –> grade is 3/4) and so on

- Select, for each service and each host category that matches the most (the one with the highest grade), vote for it by adding $+\alpha$ to the index of that host
- Add the service to the host that received most of the votes
- If the host is full (in terms of energy, memory or cpu) then start mapping the hosts with next highest votes
- Repeat the same procedure until all services are visited

5. *H-M-L Encoding Vote with Higher Priority for H Cluster:*

It is similar to the one proceeding it, however, we add $+3*\alpha$ (not $+\alpha$) to the votes of the H-category, we add $+2*\alpha$ to the M-category and $+\alpha$ to the L-category. This will allow the H-category cluster to impact the voting results more than the other two categories which will allow the H to dominate. This is done to ensure that the services that require high characteristics (cpu, energy, delay or memory) are mapped to the H-clusters.

- Randomly pick a service
- For each of the host categories (H-M-L), compare the encoding of the service categories with that of the host categories. When comparing similar categories to each other, if the numbers of the same index match, then we give $+\alpha$ to each match and so on
- Select, for each service and each category the host that matches the most (the one with the highest grade), vote for it by adding $+3*\alpha$ if

it was the H-category, $+2*\alpha$ if it was the M-category and $+\alpha$ if it was the L-category to the index of that host

- Map the service to the host that received the most votes
- If the host is full (exceeded energy, memory or cpu) then start mapping the hosts with next highest votes
- Keep on doing the same procedure until all services are visited

6. *Delay Service Selection with High to Low Distribution Over Hosts:*
   The last method was implemented to satisfy the delay factor. Here, the delay is taken into consideration when placing the service. This method is created to ensure that services are pushed to hosts that minimize the total delay factor.

   - Grade hosts according to delay and overall power. The first grade is derived by sorting the hosts according to the delay. The host that minimizes delay the most is placed at the end of the array, and so on. On the other hand, we sort (same technique of method 5) hosts according to the overall power where host that maximize overall power the most is placed at the end of the array, and so on. We considered that the given grade equals the sorted position index of that host. This means that the hosts that minimize delay and maximize overall power shall get the highest grade.
   - Randomly pick a service and map it at the host with highest grade
   - If the host is full (exceeded energy, memory or cpu) then start mapping the hosts with nest highest grades
   - Repeat the same procedure until all services are visited

At the end of the execution of the above methods, we shall generate an initial population that distributes services onto the most appropriate hosts to satisfy

the set of objectives listed in Table 1.

Method	Pushed Services	Idle Hosts	Survivability	Critical Placement	Total Delay
1	Y	Y	Y	N	N
2	Y	Y	Y	N	N
3	Y	Y	Y	Y	N
4	Y	N	Y	N	N
5	Y	N	Y	N	N
6	Y	Y	Y	N	Y

Table 1: Guarantees w.r.t Objectives

3.3 Clustered Genetic & Memetic Algorithms:

In this section we describe the Genetic Algorithm and explain how it generates populations of solutions through performing a set of operations to produce new offspring that belongs to the next generation using parents of the current one. After that, we describe the additional step that transforms the Genetic Algorithm into Memetic Algorithm. GA is an evolutionary algorithm that performs bio-inspired operations to generate generations. The process first starts by checking if the problem has a solution. In our case, a problem has a solution if and only if there exists at least one host that can manage at least one service. The GA starts by producing a randomly generated population of solutions with the below encoding format:

$$\begin{bmatrix} & Service\ 1 & Service\ 2 & Service\ 3 & \dots & Service\ s \\ Host\ 1 & 0 & 1 & 1 & \dots & 0 \\ Host\ 2 & 0 & 0 & 0 & \dots & 0 \\ Host\ 3 & 1 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ Host\ h & 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

As we observe from the above example, any service S_j can be pushed at most one time onto a fog node. We chose this option because we believe that it is better to give other services the same chance of being pushed and thus giving requesters of different services an equal chance of experiencing a good quality of service. After creating the initial random solution, the algorithm tends to fix any violations. In this work we consider the CPU and Memory factors as hard constraints, which means that any solution violating them is not acceptable. On the other hand, the other resources (delay and energy) are considered soft constraints where the quality of the solution shall degrade as they degrade. Solution K_v is considered to be violating if it hosts services that require more CPU or Memory than the fog node can provide. Violating solutions are repaired by performing two methods. The first is by moving services onto another host. The second is by removing the service completely. Now that we have an initial solution set, the GA starts by generating offspring of the existing population which represents the new generation. We use four methods to build the offspring solution. The first is the Selection method (Algorithm 9). The main aim of this method is to loop over all the solutions in the current generation (initially the randomly generation population), to select the best N number of solutions through comparing their OF (Algorithm 6), and to include those solutions in the next generation. The procedure is illustrated in Figure 4. The second method is crossover. Here we aim to randomly select two solutions out of the current population to perform a crossover procedure over them. This process is illustrated in Figure 5. The third method which allows the solution to mutate into another one is the mutation method. The aim here is to randomly select one of the solutions and to randomly select and place services on different hosts. The procedure is illustrated in Figure 6. The final method that is used in producing generations is the BiPolarSwap method. It works as a palindrome where we swap opposing poles of the randomly chosen solution. The procedure of this method is illustrated in Figure 7. The algorithm shall keep on looping until it finds the most efficient solution. This is

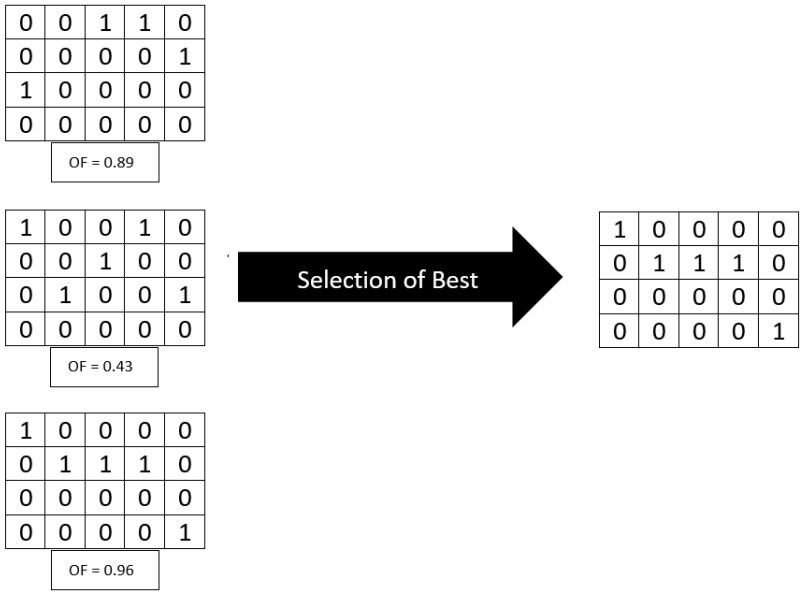


Figure 4: Selection procedure

done by either stopping after N iterations, or by stopping after reaching convergence.

Since we compare our model to the model of [25], which uses a Memetic Algorithm to solve the problem, it is worth mentioning the extra step done by the MA. As a reminder, the MA performs the same steps of the GA while adding one additional step. After applying the mentioned methods of the GA, the MA applies a probabilistic local search procedure which helps improve the individual's fitness, introduces diversity, and reduce the likelihood premature convergence [25]. In algorithm 8, if the probability is less than 0.5, we maximize the number of pushed services. On the other hand, if we have probability ≥ 0.5, we minimize the number of available volunteers. Once the MA loops over several generations and reaches a convergence in the OF value, it goes ahead and chooses the best solution from the last generation which shall be considered as the best solution for the container placement problem.

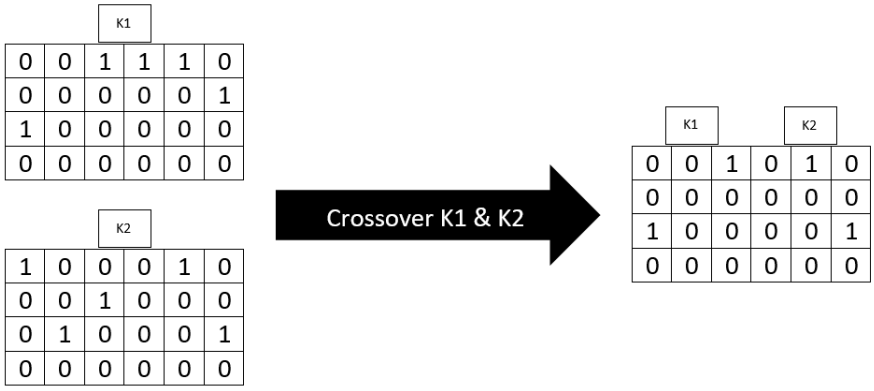


Figure 5: Crossover procedure

Figure 6: Mutation procedure

Figure 7: BiPolarSwap procedure

Algorithm 8 Probabilistic local search

Input: *RepairedRandomSolutionSet*
Output: New set of optimized solutions
Procedure Probabilistic Search
for *Each solution in the RepairedRandomSolutionSet* **do**
    if *Probability* $<$ *0.5 then* **then**
        We remove containers placed on H_i *and run them on* H_j*' if resources available are enough, and then assign any unselected service on* H_i *if resources are available after sorting them with priority level*
    if *Probability* $\geq$ *0.5 then* **then**
        We assign all services S_i *needed to available* H_i *devices depending on resources requirement, and then we discard all* H_j *and assign all services* S_i *to new set of volunteers* H_j *that can host them*

Return SelectedSolutions
=0

Algorithm 9 Selection

Input: *Population, NumberOfSolutions*
Output: New generation solutions = NumberOfSolutions
Procedure Select best solutions
for *Each solution in the Population* **do**
    if *The number of selected solution did not reach NumberOfSolutions* **then**
        Select the best solution using Algorithm 6
        Mark that the solution is taken

Return SelectedSolutions

Algorithm 10 Crossover

Input: *Population, NumberOfSolutions*
Output: New generation solutions = NumberOfSolutions
Procedure Crossover parent solutions
for *Each newly solution to be crossed* **do**
    if *The number of crossed solutions did not reach NumberOfSolutions* **then**
        Select a random solution R1 from the current population
        Select a random solution R2 from the current population
        Replace the second half of R1 by the second half of R2

Return CrossedSolutions

Algorithm 11 Mutation

Input: *Population, NumberOfSolutions*
Output: New generation solutions = NumberOfSolutions
Procedure Mutated parent solutions
for *Each newly solution to be mutated* **do**
    if *The number of mutated solutions did not reach NumberOfSolutions* **then**
        Select a random solution R1 from the population
        replace randomly the services found on H_i with that of Hj of that same solution R1

Return MutatedSolutions

Algorithm 12 BiPolarSwap

Input: *Population, NumberOfSolutions*
Output: New generation solutions = NumberOfSolutions
Procedure BiPolarSwapped parent solutions
for *Each newly solution to be BiPolarSwapped* **do**
    if *The number of BiPolarSwapped solutions did not reach NumberOfSolutions* **then**
        Select a random solution R1 from the populations
        replace the services found on the opposing poles of R1 with each other

Return BiPolarSwappedSolutions

3.3.1 Integrating Genetic Algorithm with Machine Learning:

Now that the method that allows us to create a clustered initial population is set, we can go ahead and integrate the clustering technique with the classical Genetic Algorithm. Algorithm 13 contains the same steps as of the Memetic Algorithm minus executing the probabilistic search algorithm 8. This algorithms creates generations from an intelligently customised initial population which we believe shall narrow the search space of the algorithm to the area where good solutions are found. This will allow the enhanced algorithm to generate the good solution faster than the MA and will make the quality of the solutions better, especially as the number of services and hosts increases.

Algorithm 13 Genetic Algorithm (GA) Integrated with Machine Learning

1: **Input:** *NumberOfSolutions, NumberOfGenerations, SetOfHosts, SetOfServices, SetOfUsers, w1, w2, w3, w4, w5*
2: **Output:** Best known solution between all generations
3: **Procedure** Population creation
4:
5: **if** The problem has a solution **then**
6: *InitialClusteredSolutionSet* $\longrightarrow$ *Create a clustered initial set of solution of size NumberOfSolutions*
7: *RepairedClusteredSolutionSet* $\longrightarrow$ *Repair any violation done by any solution of InitialClusteredSolutionSet*
8:
9: **for** *Each Generation* **do**
10: *NewGenerationSolutionSet* $\longrightarrow$ *Create new population using RepairedClusteredSolutionSet by Selection, Crossover, Mutation and BiPolarSwap operations described in algorithms 9, 10, 11 and 12 respectively*
11: *RepairedNewGenerationSolutionSet* $\longrightarrow$ *Repair any violation done by any solution of NewGenerationSolutionSet*
12: *RepairedClusteredSolutionSet = RepairedNewGenerationSolutionSet*
13:
14: *Return BestSolution using Algorithm 6*

3.4 Experimental Results

In this section, we perform a set of experiments to investigate the importance of the enhancement added to the GA algorithm in comparison to the Memetic Algorithm defined in [25]. Figure 8 shows the flow of comparison between the two GA methods (The enhanced GA and the Memetic Algorithm). It starts by feeding each of the methods a random/clustered initial set. The same steps are then performed on both algorithms which results in two best solutions elected by each of the methods. At the end we perform an OF comparison process to choose the best of the best.

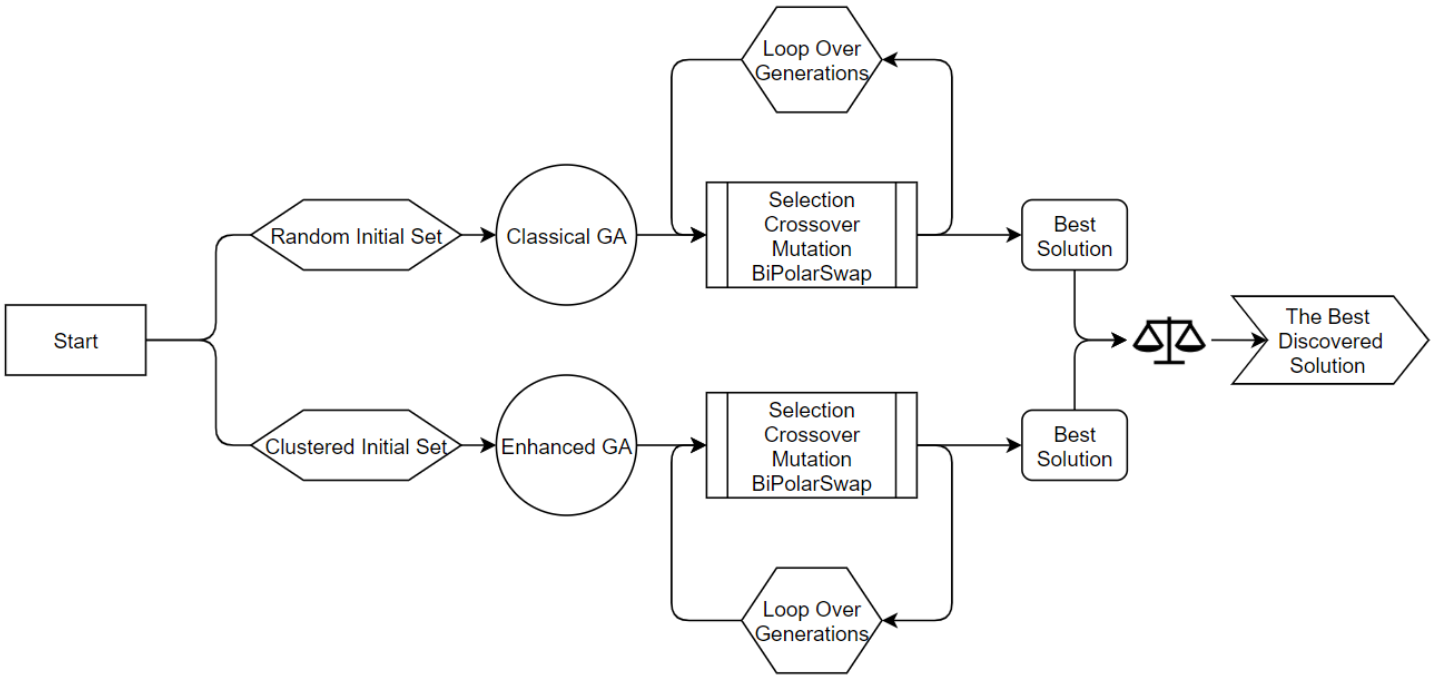


Figure 8: Flow of comparison between the classical and enhanced GA

3.4.1 Dataset Creation Details:

Since it was hard to find a dataset that contains real data including cpu, memory, energy, delay and priority information of hosts, services, and users, we had to configure our own algorithm for generating such data. This method, which takes as an inputs the number of hosts, services and users, outputs a dataset containing randomly generated information about cpu, energy, memory, delay and priority.

The range of each metric is either taken from [34] or deduced from real life examples. The details of the algorithm are presented in algorithm 14.

Algorithm 14 Dataset Creation

```
1: Input: NumberOfHosts, NumberOfServices, NumberOfUsers
2: Output: Set of Hosts, Set of Services, Set of Users
3: Procedure Data Generation
4:
5: for Each host to be created do
6:     Generate random number between 20 and 320 for cpu
7:     Generate random number between 3000 and 5000 for energy
8:     Generate random number between 1000 and 10000 for memory
9:     for Each user do
10:        Generate random number between 1 and 60 for delay
11:
12: Add each created host to the SetOfHost
13:
14: for Each service to be created do
15:     Generate random number between 5 and 35 for cpu
16:     Generate random number between 100 and 800 for energy
17:     Generate random number between 500 and 1200 for memory
18:     Generate random number between 1 and 60 for delay
19:     Generate random number between 1 and 5 for priority
20:
21: Add each created service to the SetOfServices
22:
23: for Each user to be created do
24:     for Each service do
25:         Generate random number between 0 and 1000 for the probabilistic
    rouletteValue
26:         if rouletteValue > 700 then
27:             Generate random number between 0 and 300 for requests
28:
29: Add each created user to the SetOfUsers
        =0
```

3.4.2 Quality of Solutions:

In this section we prove that the enhancement that we introduced leads to better final solutions than that of the MA. We have performed a set of experiments while varying different parameters such as number of generations, number of hosts and the number of services. Figures 9, 10, 11, and 12 test the quality of the clustered and random solutions on different numbers of generations with small number of

services and hosts. The first observation is the common initial state of the figures. The four tests show that the clustered solutions are generated in a way that helps them produce high OF values at early generations. On the contrary, the random based method starts by having lower OF values. If we look closely to the details of the figures, we can observe that the time needed for the random method to reach an OF value which is closer (but still less) to the enhanced GA is about seven generations. One might think that seven generations might not be considered a decisive factor when it comes to heuristics, which might be true. However, we should keep in mind that the number of hosts (in this case 4) and services (in this case 15) is pretty small compared to the real word scenarios. For that reason, in the next set of experiments we shall increase the number of hosts and services to simulate the real life scenarios. As the number of hosts and services increase, the problem becomes much more complex and the gap between the random and clustered methods shall expand as shown in the upcoming tests. For the current figures, we can say that the enhanced GA reached convergence before the MA suggested in [25]'s model in most of the cases. Also, the final solution produced by the enhanced version is better.

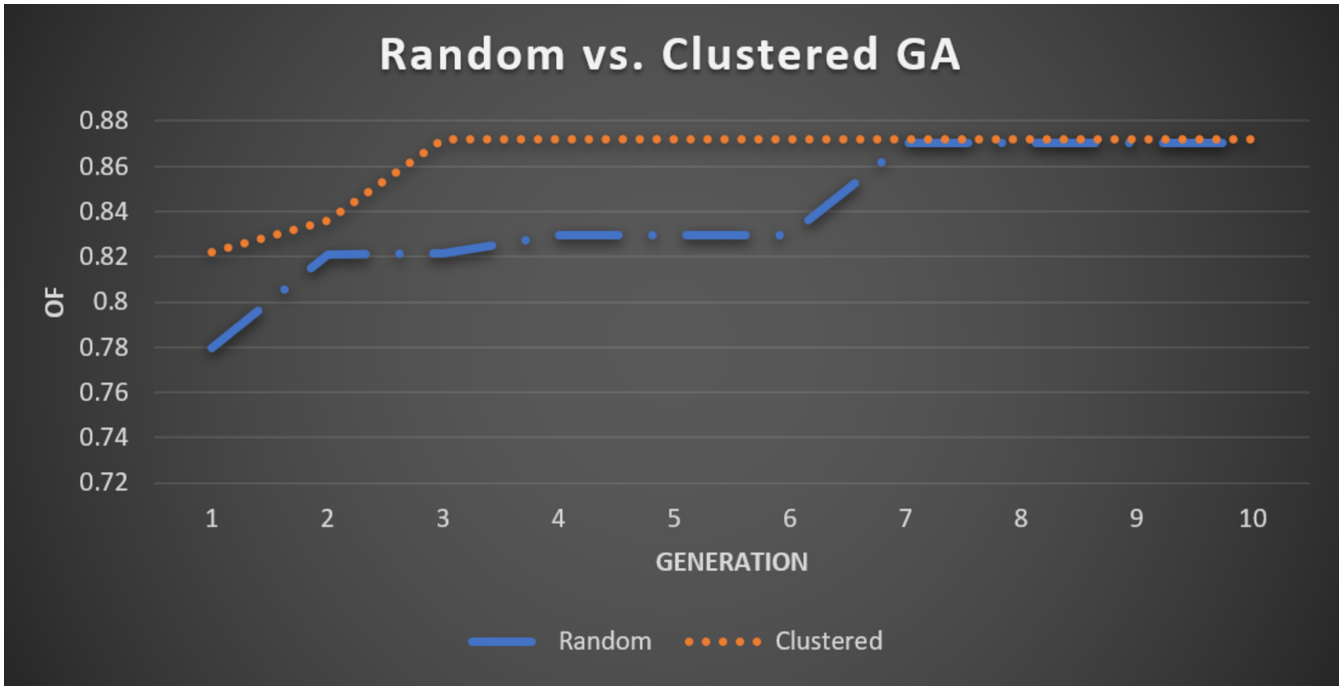


Figure 9: 10 Generations experiment including 4 Hosts and 15 Services

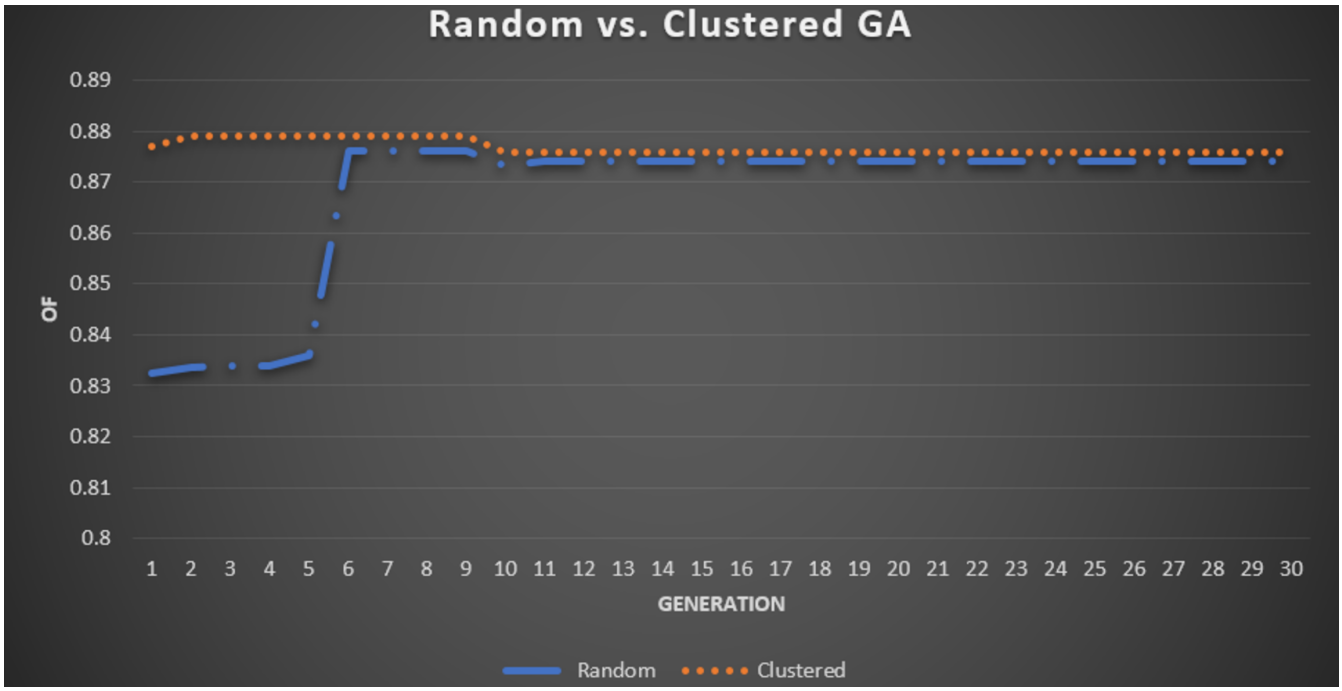


Figure 10: 30 Generations experiment including 4 Hosts and 15 Services

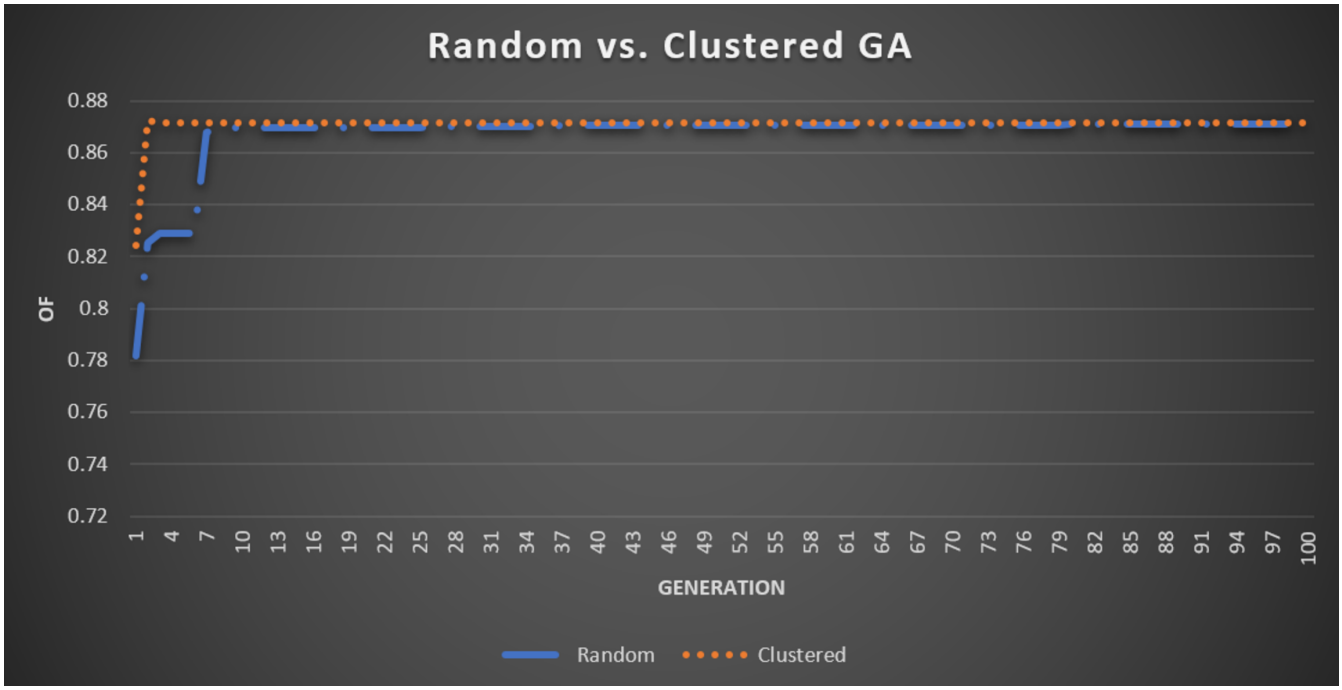


Figure 11: 100 Generations experiment including 4 Hosts and 15 Services

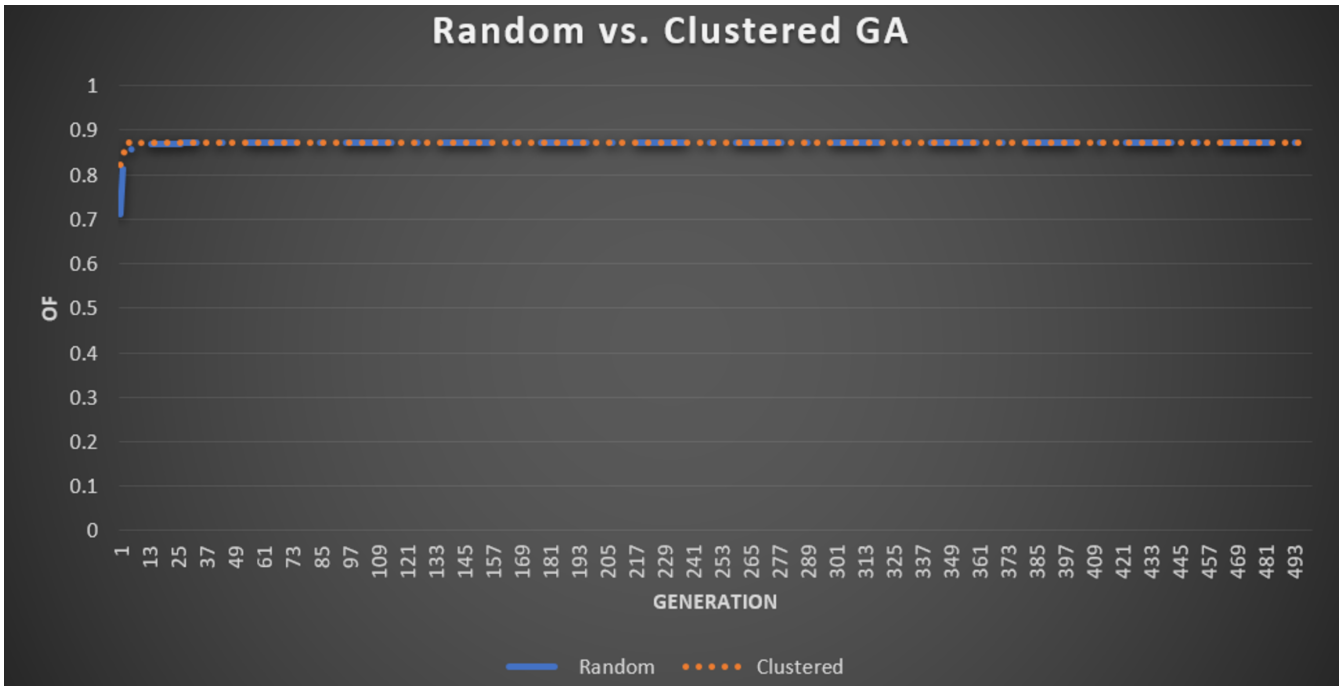


Figure 12: 500 Generations experiment including 4 Hosts and 15 Services

We now move on the second set of experiments. Figures 13, 14, 15, and 16 represent four different states of the GA where we vary the number of generations while fixing the number of hosts to 10 and the services to 30. All of the mentioned figures show a wide gap between the two methods. This is directly due to the fact that the number of hosts and services is bigger. In figure 13 we notice that the enhanced version of the GA follows a stable state after the 2nd or 3rd generation. Meanwhile, the random based MA method shows a gradual increase w.r.t the number of generations. The gradual increase that we mentioned is clearly shown in figure 12 where the GA looped over 500 generations. In all the cases, the enhanced version reached better solutions in a more efficient way since it converged to the best solution faster than the random-based MA method. This proves that clustering generated a well designed initial solution that improves the solution's quality.

We now move to the third experiment which loops over generations while dealing with 20 hosts and 80 services. Now that the problem became of a large scale, we expect to have a wider gap between the two methods. Figures 17, 18, 19, and 20 show the experimental results. A wider gap is observed as the number

of hosts and services increase. As the results of the previous figures, we notice that the MA curve tends to increase as the number of generations increases. This means that the MA is taking time to discover better solutions because the initial parent set is quite diverse and random. At the same time, the clustered curve increases with time, however this increase is minimal because the enhanced GA starts with a well constructed parent set thanks to the intelligence of the machine learning clustering approach.

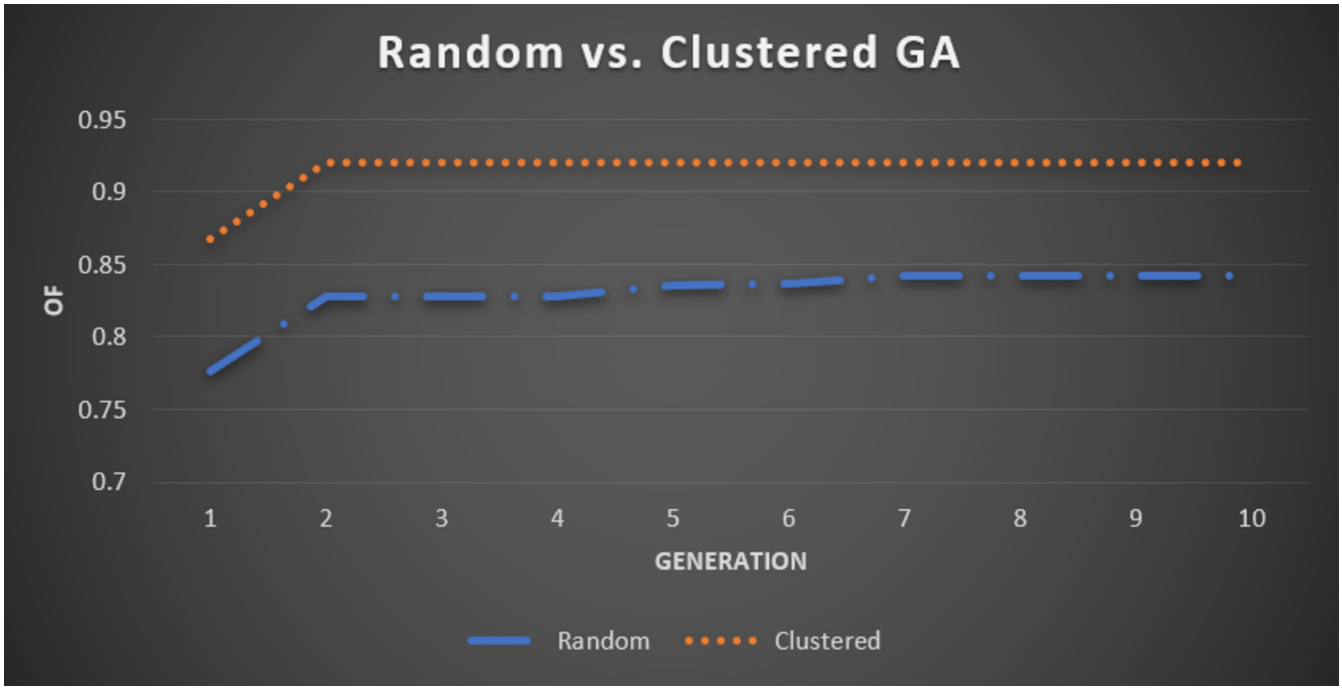


Figure 13: 10 Generations experiment including 10 Hosts and 30 Services

3.4.3 Efficiency of Solutions Convergence:

We use figure 20 as a reference for the efficiency test. As we observe from the figure, the curve of the clustered enhanced GA reaches stability before that of the MA. The clustered solution converges in a faster way towards a solution that maximizes the final objective. On the other hand, the MA approach takes a large time to find the best solution. We observe that the MA approach keeps on enhancing as the generations progress unlike the clustered curve which reaches stability at early generations. The main reason behind this is that the clustered initial set of the enhanced GA allows the algorithm to discover the best solutions

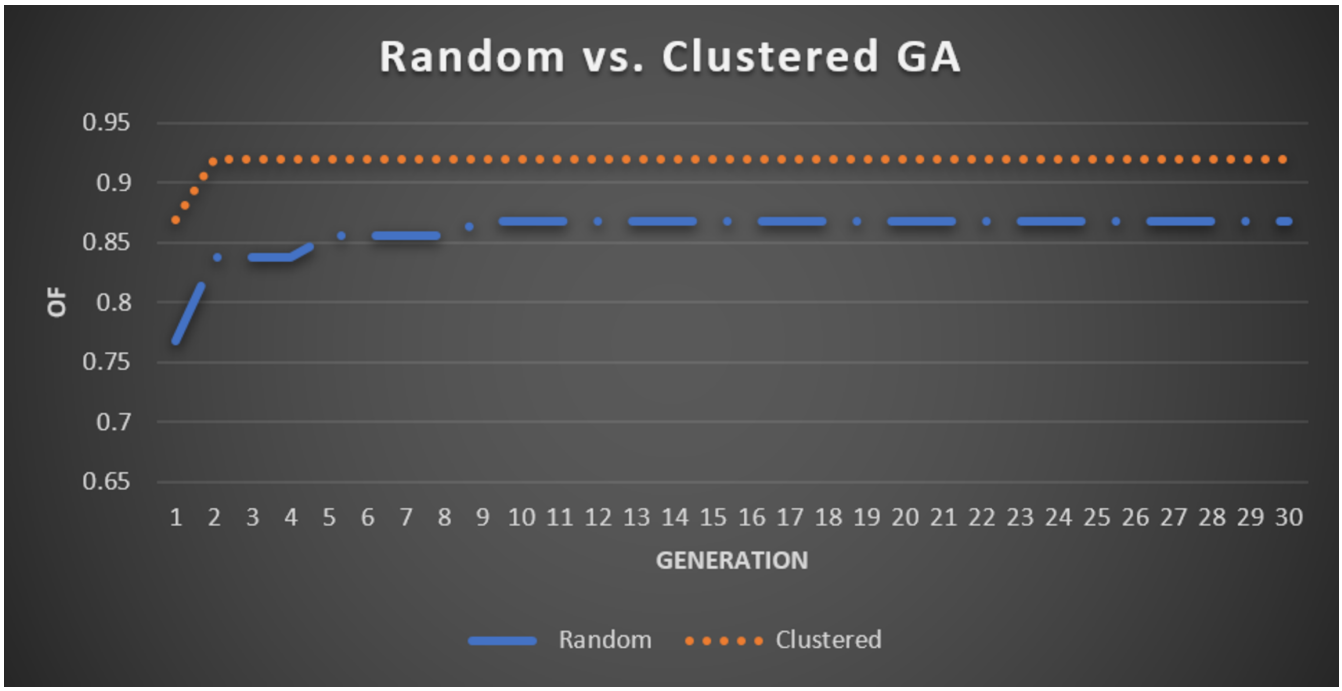


Figure 14: 30 Generations experiment including 10 Hosts and 30 Services

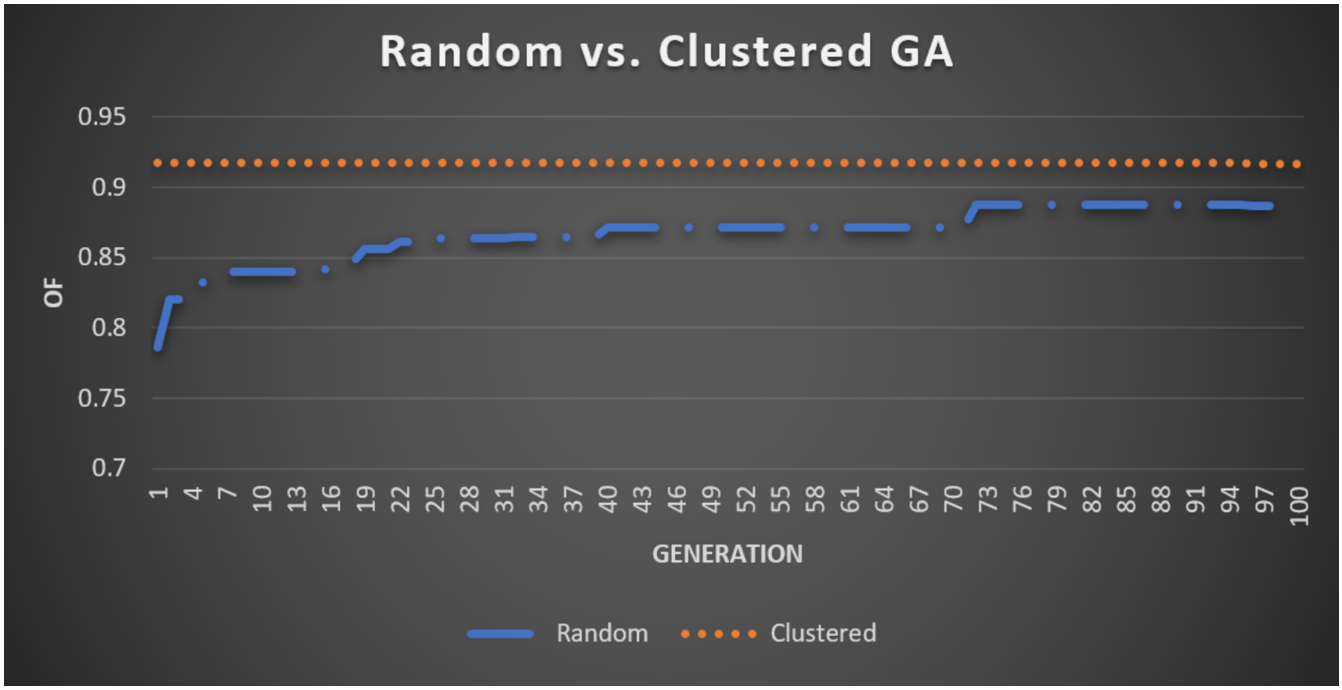


Figure 15: 100 Generations experiment including 10 Hosts and 30 Services

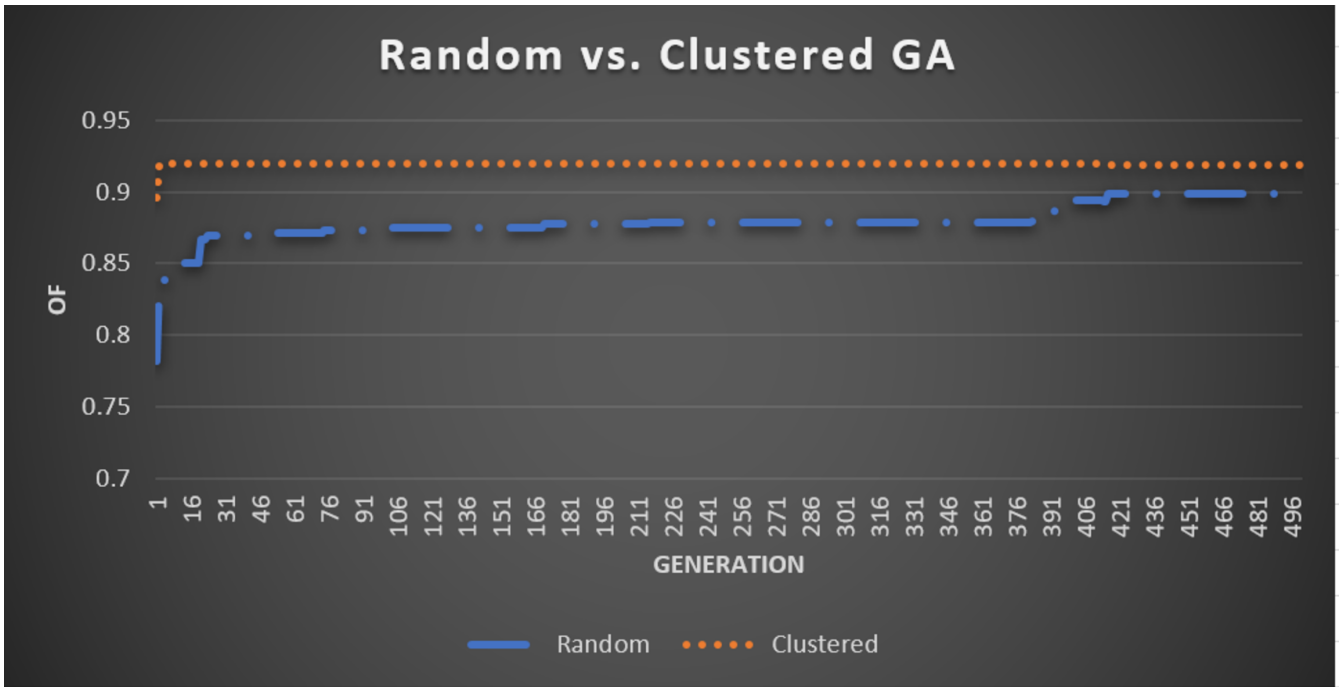


Figure 16: 500 Generations experiment including 10 Hosts and 30 Services

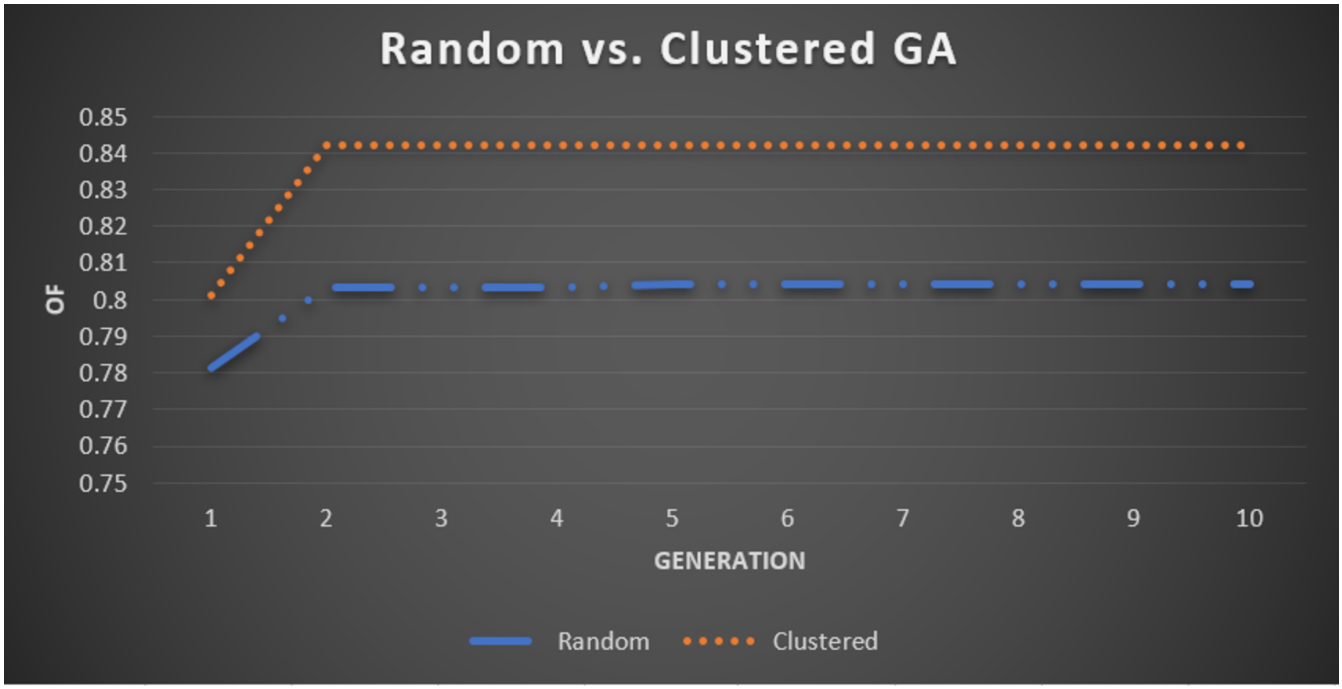


Figure 17: 10 Generations experiment including 20 Hosts and 80 Services

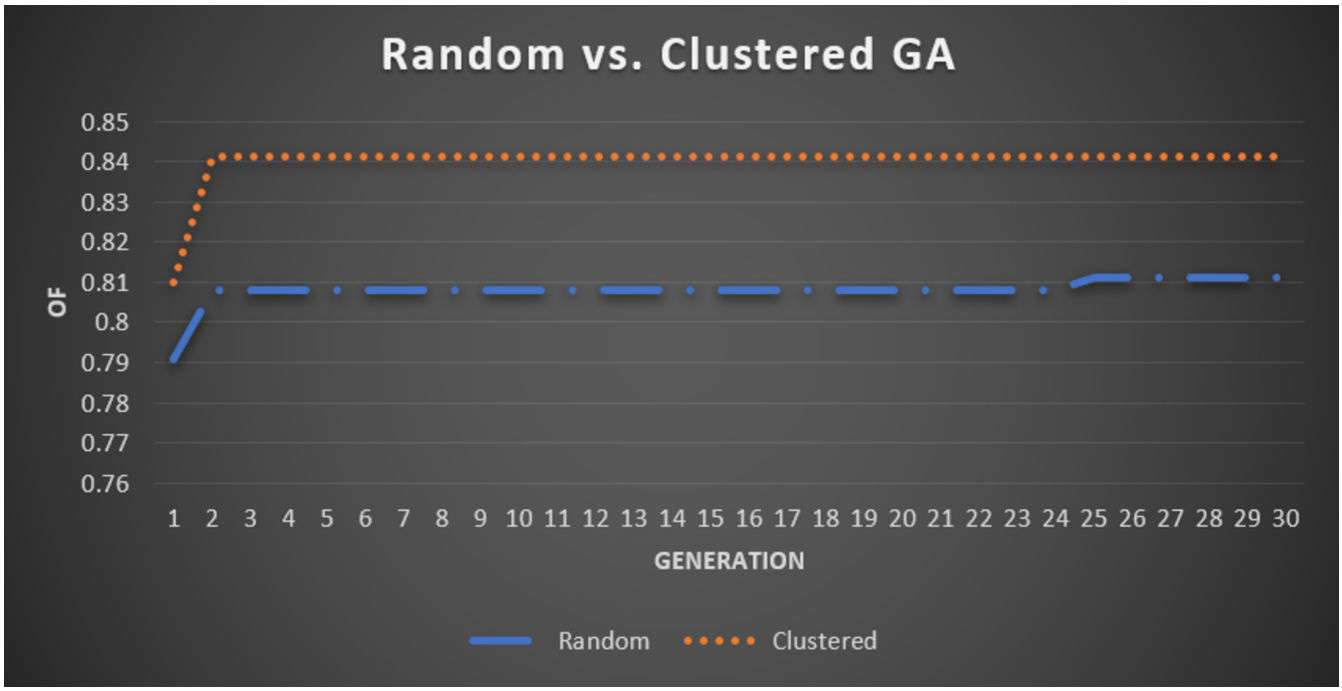


Figure 18: 30 Generations experiment including 20 Hosts and 80 Services

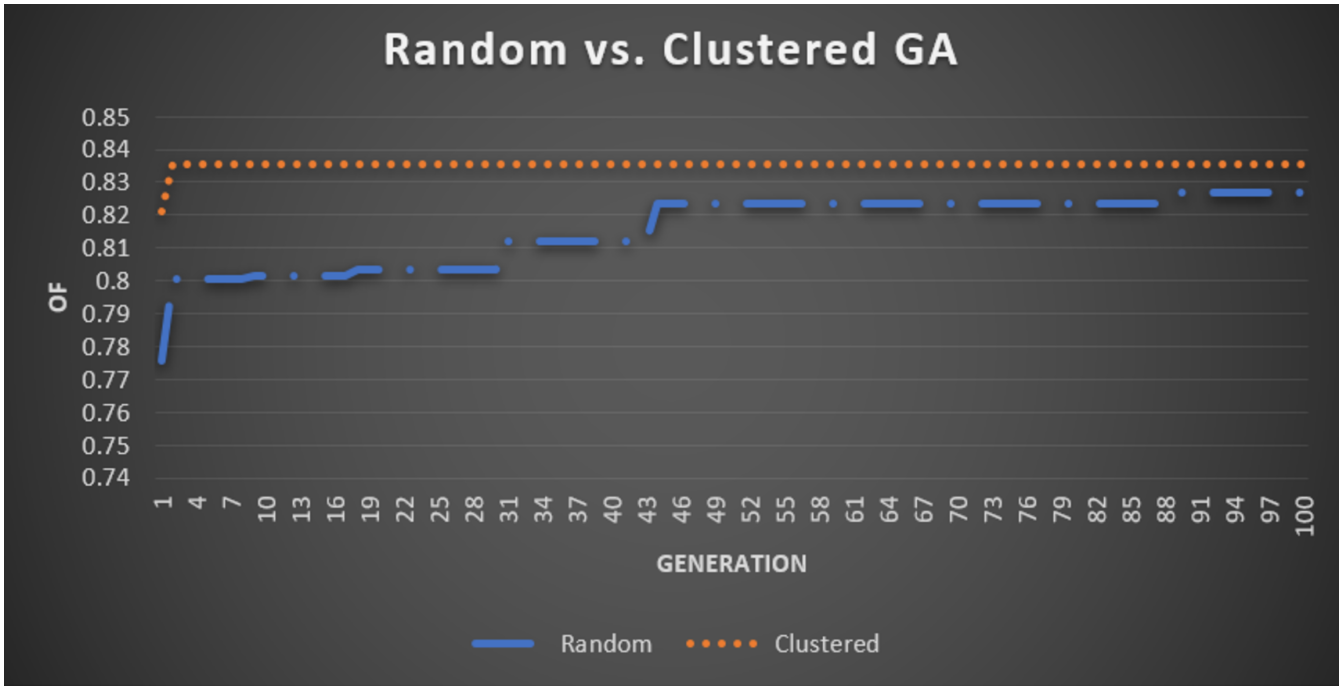


Figure 19: 100 Generations experiment including 20 Hosts and 80 Services

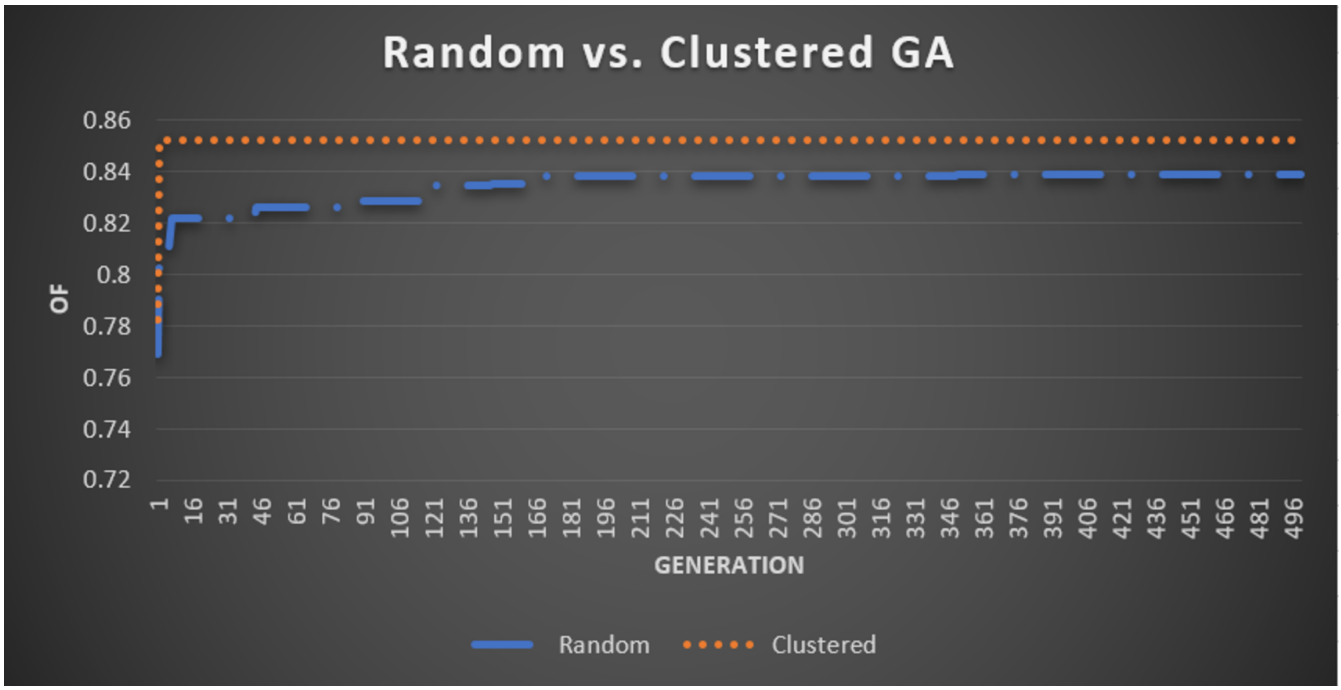


Figure 20: 500 Generations experiment including 20 Hosts and 80 Services

that maximize the OF in a faster way compared to the Memetic Algorithm, thus reaching convergence efficiently.

3.4.4 Weighted Final Solution Convergence:

In this section we test the impact of varying the weights of each objective function and show its effect on the final solution. We launch a set of scenarios to demonstrate the effect of weights on the solutions. Figure 21 shows the different considered scenarios. The first scenario targets the evenly distributed weights between the five objective functions. We notice that the idle hosts objective has the least value between the others. This is due to the fact that the other four objectives were able to reach such values just because the solution had to use more hosts to distribute the services on, which resulted in a higher overall OF value. The next case gives the priority to the idle hosts objective this time. We can clearly observe that the final solution is tailored towards this objective since it maximized it and consequently maximized the survivability factor. The third case gives both the service distribution and idle hosts objectives an equal weight which allowed the idle hosts to reach a maximum limit while keeping in mind

the other objective. The fourth and fifth cases gave priority to the total delay and to the critical placement objectives respectively. The final outcome is that the final solution maximizes the desired objective function which proves that the weights corresponding to each objective plays a crucial role while generating the final solution.

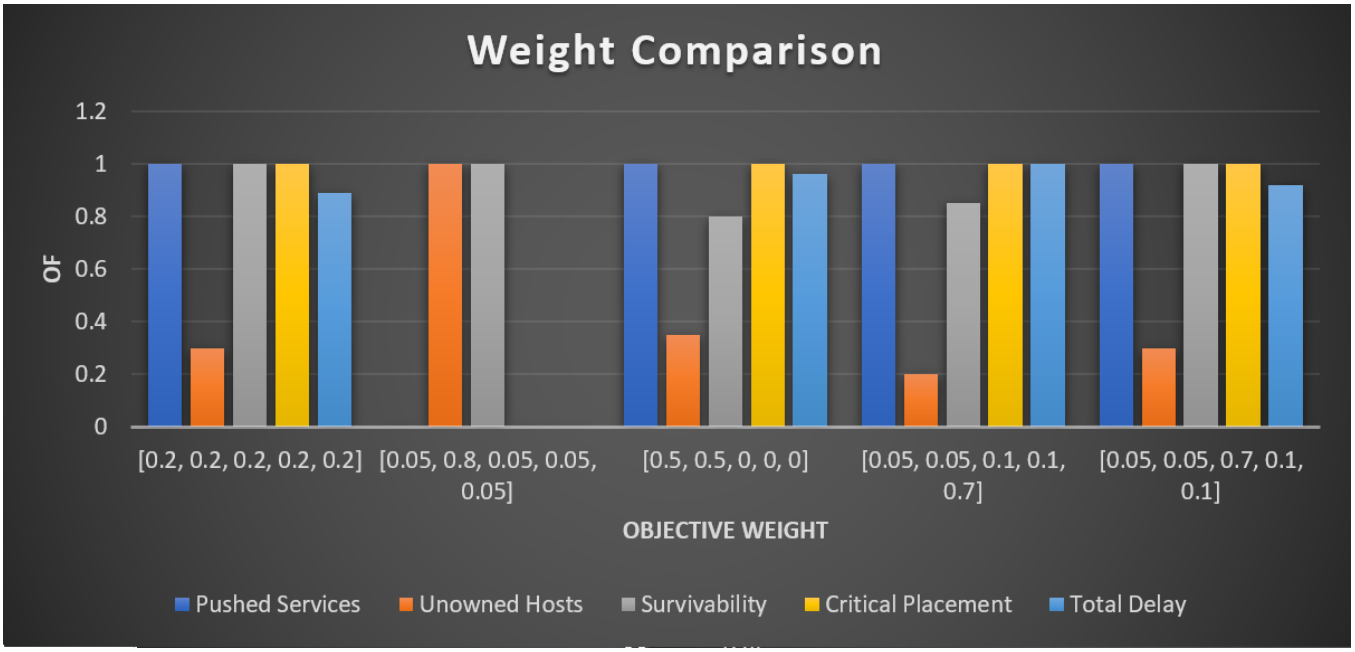


Figure 21: Comparing the effect of the weights over the final chosen solution

The results of this chapter proves that applying a clustering mechanism that uses machine learning, such as K-means, can boost pre-existing algorithms such as GA to reach better results in a more efficient way. A new set of objectives were also introduced in this chapter, compared to [25], that include more control over specifying the characteristics of the needed final solution. Also, the ability of applying actual weights to each objective was brought to live, thanks to the normalization process suggested by this work.

Chapter 4

Fog Scheduling using Machine Learning

This chapter tackles the problem of localization of on-demand fog placement. The reason behind working on such a topic comes from the need of a way that allows the creation of on-demand fogs dynamically, while keeping in mind an important factors that needs to be satisfied. Decreasing the load of requests that are being processed by the cloud is our priority. The reason behind that is the fact that decreasing the number of requests being transferred to the cloud shall decrease the network congestion, decrease the processing cost of the cloud, decrease the probability of service failure, and increase the quality of service regarding users. The problem definition, methodologies, model, and experimental work are described in the following sections.

4.1 Architecture and Methodology

In this section we introduce the architecture and methodology of our proposed approach that sums up the overall procedure of creating on-demand fogs. It is based on two different machine learning concepts and is able to decide on whether or not a fog shall be created in a certain location to best satisfies several conditions such as increasing the quality of service (QoS), decreasing the load of the cloud,

and deceasing network congestion. Figure 23 is divided into three main levels.

- The first and highest level contains the cloud. It holds three main components which are the *Services*, the *Requests Storage*, and the *Decision ML Model* which is an intelligent agent responsible of taking decisions concerning fog localization. The agent is hybrid composed of two famous machine learning concepts which are time series and reinforcement learning. The main role of this agent is to study, using the information fed by the cloud to a database (Requests Storage), the behavior of users of different locations. What is meant by "fed by the cloud" is that when a request is sent from any of the locations directly to the servers present on the cloud, those servers will take note of such an action by storing the information of the request in the Requests Storage database. Doing so, and by using techniques derived from Q-learning methodologies, the agent will be able with time to take an accurate decisions of predicting when and where a fog shall be created. An example of the requests that are saved in the storage is shown in Figure 22. These requests are made up of an IP address (which is used to determine the location of the user), a date and time, and a requested service. We will elaborate more on the request's format and usage later in this work. To provide the reader with the general idea of how things work, a brief example about the overall process is presented next. Suppose the agent learns with time that a group of users at location **L** are going to be requesting a lot of services at 4:00 pm, the agent alerts the clouds beforehand that a fog must be created at that location. This will cause the cloud to initiate a fog creation decision at location **L**.

- Level two of Figure 2 constitutes the Kubeadm cluster. Since the focus of this work is oriented towards finding the best fog localization plan, the concept of creating the Kubeadm cluster shall be briefly discussed here. You can refer to [25] for more details about the dynamically created clusters. In our previous work, we have suggested a new concept of creating on-demand

```
199.72.81.55    [01/Jul/1995:00:00:01   -0400]  GET /history/apollo/ HTTP/1.0   200 6245
199.72.81.55    [01/Jul/1995:00:00:01   -0400]  GET /history/apollo/ HTTP/1.0   200 6245
199.120.110.21  [01/Jul/1995:00:00:09   -0400]  GET /shuttle/missions/sts-73/mission-sts-73.html HTTP/1.0   200 4085
199.120.110.21  [01/Jul/1995:00:00:11   -0400]  GET /shuttle/missions/sts-73/sts-73-patch-small.gif HTTP/1.0    200 4179
205.212.115.106 [01/Jul/1995:00:00:12   -0400]  GET /shuttle/countdown/countdown.html HTTP/1.0  200 3985
129.94.144.152  [01/Jul/1995:00:00:13   -0400]  GET / HTTP/1.0  200 7074
129.94.144.152  [01/Jul/1995:00:00:17   -0400]  GET /images/ksclogo-medium.gif HTTP/1.0 304 0
199.120.110.21  [01/Jul/1995:00:00:17   -0400]  GET /images/launch-logo.gif HTTP/1.0    200 1713
205.189.154.54  [01/Jul/1995:00:00:24   -0400]  GET /shuttle/countdown/ HTTP/1.0    200 3985
205.189.154.54  [01/Jul/1995:00:00:29   -0400]  GET /shuttle/countdown/count.gif HTTP/1.0   200 40310
205.189.154.54  [01/Jul/1995:00:00:40   -0400]  GET /images/NASA-logosmall.gif HTTP/1.0 200 786
205.189.154.54  [01/Jul/1995:00:00:41   -0400]  GET /images/KSC-logosmall.gif HTTP/1.0  200 1204
199.72.81.55    [01/Jul/1995:00:00:59   -0400]  GET /history/ HTTP/1.0  200 1382
205.189.154.54  [01/Jul/1995:00:01:06   -0400]  GET /cgi-bin/imagemap/countdown?99,176 HTTP/1.0 302 110
205.189.154.54  [01/Jul/1995:00:01:08   -0400]  GET /shuttle/missions/sts-71/images/images.html HTTP/1.0    200 7634
```

Figure 22: A sample of the requests captured by the NASA server [1]

Kubeadm clusters which aims to serve users. For myriad number of reasons, an architecture that combines both containerization and micro-service technologies were used. Having this said, our cluster architecture was constituted of a master node representing an orchestrator and a worker node. The master is responsible for creating the cluster, adding and removing fog nodes, and monitoring the status and services running on the different volunteering worker nodes. On the other hand, worker nodes, which are chosen by the master node, are responsible for deploying the services and serving users. As a relation to what we have mentioned above, Kubeadm clusters shall be created once the cloud initiates the request of constructing fogs at a given location **L** containing all demanded services. Our previous work never took into consideration a way of scheduling clusters to best serve both users and the cloud. On the contrary, this work serves as an extension to increase the performance of the overall cloud-fog model.

- The third and final level contains the users or devices requesting the services from the cloud. Any newly connected device that is trying to use a service shall be initially routed directly to that cloud. If any existing Kubeadm clusters are near that device, the cloud will rout that device to the nearest cluster to obtain the service. On the other hand, if no near clusters are available, the cloud will serve the user and save the request. If it happened that the number of requests deserves the creation of a near cluster according

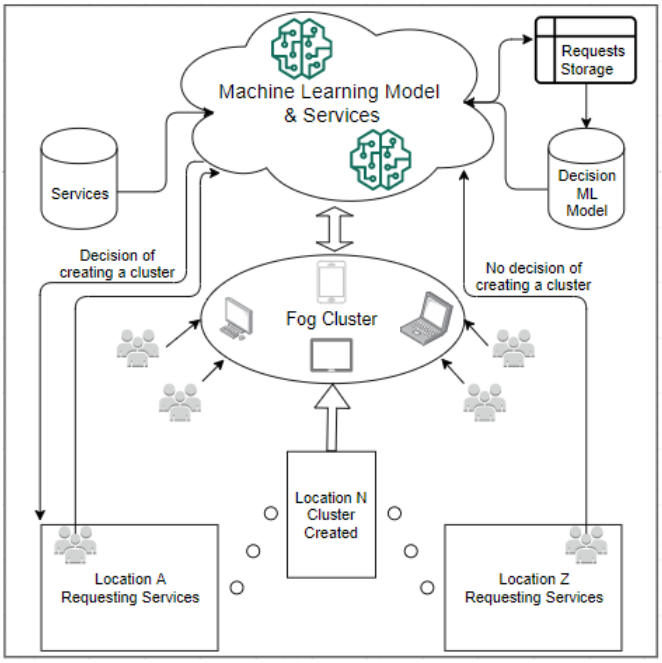


Figure 23: Fog Scheduling Model

to the behavioural learning of the agent, the cloud will initiate a request for constructing one. This is shown as an example of location **A** in Figure 23. Otherwise, and as shown at location **Z** in Figure 23, if the agent finds that the intensity and number of incoming requests might not best satisfy the conditions of creating a fog, it will discard the creation decision.

After presenting the three main levels of the model, it is time to discuss the reasoning and motivation behind our choices. Level one is the brain of the model. It contains the agent, services and the requests database. The reason behind placing those three component in the highest level is to make sure that the agent has a complete view of all requests taking place. This will allow it to make appropriate localization decisions. The only way to do so is by placing those components on a high level where all initial requests are received and where fog updates are delivered. As shown in Figure 23, there is a link between the cloud and the fog cluster. Since all requests after deploying a fog gets processed by that fog (i.e. no requests will be sent to the cloud anymore), the fog's master node must then send periodic updates to the cloud containing details about the requests received. After doing

so, the cloud can store those details in the Request Storage. In the second level, the motivation behind deploying Kubeadm clusters that use containerization and micro-service technologies is to reduce the resource consumption of fog devices. Containers use the operating system of the hosting device which allows less resource consumption compared to virtual machines that create an independent environment having independent operating systems on hosting device. Also, our fog cluster uses micro-services which are lightweight versions of the services. This will also decrease the resource usage of the hosting devices. More information about this is found in [25]. In general, the motivation behind this architecture is that it allows an overall monitoring of the whole topology which guarantees accurate predictions while ensuring an adequate resource consumption of hosting devices.

4.2 Fog Localization Process

In this section we present a new approach using two machine learning concepts for setting a better fog localization plan. In the previously used on-demand fog formation architecture, the decision of creating a fog was based on the number of incoming requests from a certain location. If that number exceeded a predefined random threshold, the cloud sends a request for creating a fog near that location to serve those incoming requests. However, after performing some experiments, the threshold-based approach turned out to have minimal effect on decreasing the number of requests reaching the cloud. This was due to the dynamic fluctuating behavior of the requests throughout the hours of the day. Since the previously targeted topic (creating fogs on-demand anywhere and at any time) does not set any boundaries on the number of available fogs to use, we decided to zoom in from the expanded definition to specify limits and have a better and more realistic scenario to perform our experiments on. Since having infinitely available fogs in all locations is most unlikely to happen in real life, we specify a constant number of fogs **M** that can be used for serving locations. Also, we decided to

add a punishment value **T** representing the time for reserving the fog's resources. After exceeding that time **T**, those resources get freed and are ready to serve another location selected by the localization plan method used. It is important to note that the punishment value has nothing to do with the reward added to the Q-table. This value is used to limit randomness and is directly linked to the cost of creating a fog. Higher punishment value is equivalent to higher fog creation cost.

Problem Definition: Given **M** number of fogs that can be created at **L** locations, predict the best possible distribution plan that can maximize the quality of service (QoS) for the biggest number of clients, maximize the number of processed requests by the fogs, minimize the processing load of the cloud, and minimize the network congestion. We can clearly see from the above definition that **M** is a finite value while **L** is an infinite one. Consequently, and as mentioned in the abstract, our model uses a technique derived from Q-learning rather than implementing the original Q-learning technique. It sets extra conditions to the well-known original Q-learning technique to make it adapt to the infinite number of locations that might contact the cloud at any hour of any day. Additional information about this idea is provided in section 4.2.
In this problem definition, we have some goals to accomplish.

1. Maximize QoS: This occurs when the fogs are deployed in locations where a lot of devices are requesting services. It is known that when the service gets closer to the client, the QoS increases [9]. Initially, that was the motivation behind the fog concept. In our model, we maximize QoS by making sure that the maximum number of clients are served by fogs rather than clouds.

2. Minimize the cloud's processing load and network congestion: by maximizing the number of processed requests by the fogs, the cloud shall have lower number of requests to worry about. By doing so, the network congestion shall decrease (because less requests are being sent to the cloud) and the

processing load shall minimize (less requests to process).

In general, all of the above goals are fulfilled by satisfying one objective which is deploying the fog in the best location to serve the largest number of requests. This is the main motivation behind creating our machine learning model. We shall now present the different approaches used in today's industry for solving the fog localization problem and then move on to discuss our proposed model.

4.2.1 Threshold-based Approach

In this approach, a random threshold is obtained. If the number of requests received from a specific location exceeds that threshold, one of the **M** fogs would be used to serve that location. Allocating the resources of a fog to a location means that this fog cannot serve any other locations unless the punishment time **T** is passed. As long as that allocated fog is receiving requests, **T** will be refreshed to its initial given value. For example, let us suppose that **T** is 6 hours. We allocate the resources of fog **X** to location **L** at time t = 0. As long as the fog is receiving requests from that location, **T** will be constantly refreshed to 6. Suppose at t = 5 the fog stopped receiving requests. This means that **T** will be decremented by 1. This process keeps on happening until **T** reaches 0. When this condition occurs, the resources of that fog will be freed and are labeled as ready to be used by another location (or maybe the same one if it received the needed number of requests again). It is important to note here that this approach focuses on exploiting the locations generating the biggest amount of requests. However, this might not always lead to the best needed solution since the behaviour of requests might change with respect to time and date.

4.2.2 Random-based Approach

This approach as the name states follows a random flow of fog-location distribution. Here, after checking all the locations that established a connection with the

cloud, a randomly selected location at a random hour of the day is chosen to have a serving fog. This approach, unlike the threshold-based, focuses on only exploring the environment by randomly selecting locations. If the punishment value is small, this approach would serve well since it would be trying all possibilities at different times with minimal cost. To elaborate more on this point, suppose a fog **X** serves location **L** with a punishment value equals to 1 hr. If we are using the random-based method, the model will randomly choose locations and set fogs to serve requests. Since the punishment value has a small value, the model will be able to cover most of the locations because it would be simply switching between locations with a small punishment cost to pay. This is not applicable in real life scenarios where the punishment value is high. For such a reason, we consider in our experiments adequate punishment values which mimic real life scenarios.

4.2.3 Machine Learning Approach

This approach which is a hybrid between two machine learning concepts, Time-Series & Reinforcement learning, uses a Q-table that allows a prediction with high success percentage. This method focuses on both exploration (by introducing some randomness in choosing locations) and exploitation (by selecting the locations with highest Q-table values) while gathering information about the environment. As time progresses, the model learns how end users behave towards services on the cloud, the times at which services are requested and the intensity of incoming requests with respect to date and time. All of this can be achieved by periodically updating the Q-table until the internal results converge to an adequate solution. Below is a representation of the Q-table format.

$$\begin{bmatrix} & Loc\ A & Loc\ B & Loc\ C & \dots \\ Hr\ 0 & - & - & - & \dots \\ Hr\ 1 & - & - & - & \dots \\ Hr\ 2 & - & - & - & \dots \\ \dots & \dots & \dots & \dots & \dots \\ Hr\ 23 & - & - & - & \dots \end{bmatrix}$$

Basically, our Q-table is a two dimensional array containing a list of locations representing the origin of the received requests as columns and a list of hours representing the hours of a day as rows. At the beginning of execution, and since this algorithm follows the reinforcement methodology, the initial values of the Q-table at each hr-location are set to zero and get updated with time according to a Q-table updating and reward formulas. Since locations are not initially known, the algorithm listens to incoming requests and dynamically adds locations as new columns in the Q-table every time a new location is detected. By applying this approach, our model would be able to adapt to any new incoming locations. The bigger the value in the Q-table cell, the higher the probability of creating a fog at the selected cell's location and time.

Original Vs. Derived Q-learning

Q-learning is a process that allows selecting the best action in a given state. Five main keywords are used when discussing this topic. We start with the *agent*. An agent is a component that takes decisions in making certain actions. Each *action* performed yields a state. A *state* is defined by certain characteristics of the *environment* in a given situation. According to the current state, the agent decides on the next action to perform. What actually happens in this process is that the agent decides on an action which directs the current state S_1 to another state S_2. This action may be good or bad. If it is a good action, a positive reward

is marked in a Q-table, or else a negative reward is given. A *reward* is simply a value that is either added or subtracted for the Q-table. An example shall be given to clarify the concept of the original Q-learning.

Suppose we have a robot that is trying to learn how to get from its current position to another targeted position. The environment is similar to a chessboard where each square has coordinates and is given a reward value. The reward is -10 if the robot steps into a square containing a wall, +1 if it is empty and +50 if the robot reaches the target. In this scenario, the robot is the agent that is taking decisions. As we can see here, the environment contains a limited number of states (the squares) and a limited number of actions (forward, backwards, left, or right). What will happen is that the model will explore the environment by initially taking random actions and marking the rewards in the Q-table. In the next iterations, the robot shall be relocated to its initial position and will depend on the data in the Q-table to take the new set of actions. Due to the fact that the environment is static and nothing is changing while the learning procedure is happening, we can safely use the above suggested process which guarantees good results. However, suppose that at each iteration the position of the obstacles changes and the chessboard environment expands to an unknown length. How would it be possible to create a chain of knowledge knowing that everything in the environment is changing using the original Q-learning?

The problem that we are trying to solve in this work contains infinite number of locations that can be added at any time of the day. We can link this to the unknown expansion of the chessboard. Furthermore, the number of incoming requests from each location might also vary. This can be linked to the change in obstacle positions. What we are trying to show here is that it is not applicable to use the original Q-learning as is. As an answer to the above question, a modified version of the original Q-learning which adapts to modifications and changes in

the environment must be modeled to achieve the desired learning. A mapping between our Q-learning problem and the original Q-learning scenario shall be presented next.

The agent that takes decisions is located in the first level of Figure 23. In our case, this agent performs actions which are defined by creating a fogs at locations **L**. The states in our case are the hours of the day. An action of creating a fog at location **L** is directly linked to the current state. To periodically update and insert rewards in the Q-table, we formulated equations that serve in the periodic update of locations (including new locations) and in the addition of rewards where fogs are located. It is true that we only apply positive rewards, however, the reward increases with respect to the number of processed requests by the fog. For example, suppose that the total number of incoming requests is 10. If *fog1* serves 8 out of the 10 while another *fog2* serves the other 2 requests, the reward given to *fog1* shall be +0.8 while the reward given to *fog2* shall be +0.2. As a summary of the above process, the agent performs periodic updates on the Q-table where new locations are inserted and old locations are updated according to the number of received requests. It then takes decisions in fog placement. The agent then calculates the reward for each deployed fog and adds it to the Q-table. Finally, the next decision of creating fogs is derived from the Q-table where the biggest value is chosen (-hr-loc-).

Q-table Formulas

1. The updating formula for the Q-table suggested by this work takes into consideration three main elements. The first is called Intensity. This element describes how fast requests are being sent from location L with respect to time. Intensity can help identify how frequent the services are being used from a specified location at a specified state. The second element is NoR (Number of Requests). It indicates the number of requests being served by

a fog at location L. TNoR on the other hand, which is the third element, represents the total number of requests that are served on all fogs at all locations. The calculation of both NoR and TNoR is done with respect to t which represents the current state. This formula will guarantee periodic updates to the Q-table. It makes sure that the internal values of the Q-table advance with respect to the number of requests sent from the end users at each state. It also ensures dynamic insertions of new locations while giving them the appropriate Q-table value depending on their Intensity 4.1, NoR 4.2 and TNoR 4.3.

- *Intensity Formula:* The output represents the intensity of incoming requests. Intensity illustrates how far apart the incoming requests are being received with respect to time. As the gap between the incoming requests from a certain location increases, the difference between the newly received request and the one preceding it will increase. This will increase the intensity value. In other words, the bigger the intensity value is, the further the requests are. Such a value is integrated within the overall updating formula since it can capture the behaviour of requests at a given point in time. For instance, at t = 7, if a lot of requests are being sent in a small period of time (i.e. the intensity value decreased) $\implies$ we might be needing a fog for that location at that given point of time since many requests within a small period of time are being received.

$$Intensity\ (t,L) = \frac{\sum_{r=1}^{R_L^t} T_{r+1} - T_r}{N} \tag{4.1}$$

Where:

  - t: Is the hour of the day (state).
  - L: Is the Location.

- R_L^t: Represents the requests captured at time t from location L.
- T_r: Represents the time of the received request r.
- N: Is the number of requests at state t.

- *Number of Requests Formula:* Using this, we can extract the total number of requests received from a specific location L at a given hour. This helps in building up the final updating formula. By extracting this value, we can set a ratio to differentiate between the amount of incoming request of different locations. This will result in an increased value in locations processing high number of requests.

$$NoR(t,L) = \sum_{r=1}^{|R_L^t|} NoR + 1 \tag{4.2}$$

Where:

- t: Is the hour of the day (state).
- L: Is the Location.
- R_L^t: Represents the set of requests captured at time t from location L.

- *Total Number of Requests:* This value is calculated by adding all the incoming requests from all locations at a specified t. In order to create the ratio that we previously talked about, we should divide the NoR by the total number of incoming requests (TNoR). Doing so will reveal the percentage of requests sent w.r.t other locations.

$$TNoR(t) = \sum_{l=1}^{L} \sum_{r=1}^{|R_l^t|} TNoR + 1 \tag{4.3}$$

Where:

- L: Represents the Locations.
- R_L^t: Represents the set of requests captured at time t from locations L.

- *Q-table Updating Formula:* The updating formula is a composition of the above listed formulas. It aims to increase the value of the Q-table in the cells containing high number of requests and having a small intensity interval. Also, when new locations are discovered, this equation shall be used to directly update their Q-table values to adequate ones.

$$Updated\ Value(t,L) = Q_Old_tL + (\frac{\alpha NoR(t,L)}{TNoR(t) + Intensity(t,L)}) \quad (4.4)$$

Where:

- Q_Old_tl: Represents the old value in the Q-table at cell hr-location.

The updated value will replace the old Q-table value at the hour-location cell. Because of the nature of such an equation, the locations that are sending high number of requests with low intensity intervals will receive a larger addition on their old Q-table values which makes them better candidates for the fog creation process at the specified hour of the day. In general, if the number of processed requests at fog f increases, the output of the updating formula gets larger and thus increases the probability of deploying a fog at f's location in the same future states. Note that the value of α in the above equation is a con-

stant that aims to give higher values for locations that process more requests. The purpose of this constant is to increase the impact of new incoming locations since they would only contain a zero as an initial value. In order for them to compete with other locations, we should pump the updating value. If we increase this constant, the newly registered locations will dominate more rapidly compared to having a smaller constant (only if the number of requests is high of course). In our case, this constant equals 2.

Algorithm 15 Machine Learning for Fog-to-Location Creation Prediction

Input: Set of Incoming Requests
Result: Best Locations for Fog Creation
Procedure PREDICTION
I = 1, N = 1, M = Number of available fogs
While *True*:
if *Day* = 1 **then**
    for *Each Hour* **do**
        Gather incoming requests from different locations
        Create fogs at random locations (exploration) without exceeding M fogs per day
        At the end of the hour:
        Update the values of each location (new or old) in the Q-table using Eq. 4.4
        Add the reward of Eq. 4.5 to the locations hosting fogs
if *Day* > 1 **then**
    Get the highest N values from the Q-table (-Hr-Loc-)
    Create N fogs at the specified hours & locations
    Create M - N fogs at random locations and states over the whole day
    for *Each Hour* **do**
        Update the values of each location (new or old) in the Q-table at current hour using Eq. 4.4
        Add the reward of Eq. 4.5 to the locations hosting fogs
    if *N* < *M* **then**
        Increment N by I

2. The reward formula is the ratio that we have mentioned earlier. It is made up of both equations 4.2 and 4.3. A reward is added to the Q-table at a specified location and state only if a fog is deployed there. Similar to

the original Q-learning concept, we intent to reward the hr-loc cell in the Q-table by adding to it the ratio using 4.5

$$Reward\ (t,L) = \frac{NoR(t,L)}{TNoR(t)} \tag{4.5}$$

Where:

- t: Is the hour of the day (state).
- L: Is the Location.

Given a predefined number of available fogs, the machine learning algorithm expressed in algorithm 15 tends to learn the behaviour of the incoming requests using their source location and their time-stamp in order to have a -Fog-Loc-prediction methodology with high success percentage. By having a better success percentage we mean a better prediction of where a fog should be deployed to better enhance the QoS of the end users, decrease the processing pressure performed on the cloud, and decrease the overall network congestion. Algorithm 15 starts at Day = 1 by gathering incoming requests to the server. At first, the Q-table is simply empty since the algorithm has no information regarding any location. When the requests start appearing, the algorithm checks the source location of the incoming requests and adds them to the Q-table at that specific state and gives it an initial value which is equal to zero. At the end of that hour, and after capturing all incoming requests to the cloud, the algorithm performs some statistical calculations regarding the effectiveness of placing fogs at the different locations of that state. In this step we do not place an actual fog on the locations but rather we calculate statistics of what could have happened in case we added one there. It is important to note that no rewards shall be given to locations where no actual fogs are placed. The calculation of the values is done by performing the Updating Formula of Eq. 4.4 on the different locations of the state.

On the other hand, the cloud would be creating random fogs that are actually deployed in certain locations. In addition to the updating value that is added to those locations, an extra reward is also given using Eq. 4.5. The algorithm keeps on doing that until the first day is over. Because of the complete randomness in the first day, this phase is called an exploration phase. Before starting with the second day, all of the resources get freed. At Day = 2, the algorithm starts depending a little more on the information and statistics captured from day 1. At the beginning of that day, the algorithm searches for the cells containing the highest N values in the Q-table. By doing so, we would be selecting the best fog creation locations that better served our aim in day one. It it quite important to keep some randomness in case we have better solutions in the zone, and for that the other M - N fogs are randomly distributed over that day. As a reminder, M is the total number of available fogs, N is the number of specified -Hr-Loc- and I is a constant that decides the speed of transitioning from the exploration phase to the exportation phase. The maximum number that N can reach is M and that is when the algorithm starts pure exploitation.

After selecting the best cells and choosing other random cells, the algorithm creates fogs at those locations. At the end of each hour, the algorithm updates the values of the Q-table by using Eq. 4.4 and adds rewards to the locations states where actual fogs are deployed using 4.5. This keeps on happening until the end of the day is reached. When this happens and before moving on to day 3, the value of N gets incremented by 1 if N is less than M. On Day = 3, the algorithm performs the same procedure as of day 2, but this time, the randomness is decreased by increasing N. With time, the agent starts forming an idea about the behavior of the incoming requests and then it shall be able to perform accurate predictions regarding fog placement. This is all possible using the values inside the Q-table. It is important to note that within the same day, a fog can be allocated to different locations. However, this depends on a value that we have

mentioned which is called the punishment value. As a reminder, a punishment value represents how many hours a fog with no more incoming request shall stay in the same location. If within the same period that fog receives new requests, the punishment value shall be refreshed. When the value reaches 0, the resources of the allocated fog shall be released.

4.3 Experimental Results

In this section we compare our Machine Learning model to the randomness and threshold based approaches used in the literature [39][17][27]. [39] used time base threshold scheduling decision, therefore we consider it as an example of the **threshold** scheduling decision. On the other hand, [17][27] did not consider the time factor in scheduling task placement on fogs, therefore we consider it in this work as **random** time task scheduling. Our input list constitutes of a set of requests captured on a Nasa server over 14 days [1]. Each request contains a source IP address, a time-stamp, a link to the requested service, and a number representing the size of the request as shown in Figure 22. Locations are divided according to the first set of numbers before the first dot of the IP address. All requests having the same initial numbers must belong to the same location. Each of the above methods have been implemented and tested while having the same predefined conditions. The code has been written using Python language and the results of 223425 differed incoming requests to the Nasa server, which represent over 65 different locations, shall be shown and explained below.

The figures showing the experiments results are presenting the number of requests being served by the cloud on different approaches. Therefore, the aim is to minimize the load of the cloud by minimizing the number of request served by it. This implies that better service scheduling on fog yields to less number of requests served by the cloud, therefore less load.

In the first experiment, we set the total number of available serving fogs to 20. Three different sub-experiments were performed to show the positive effect of

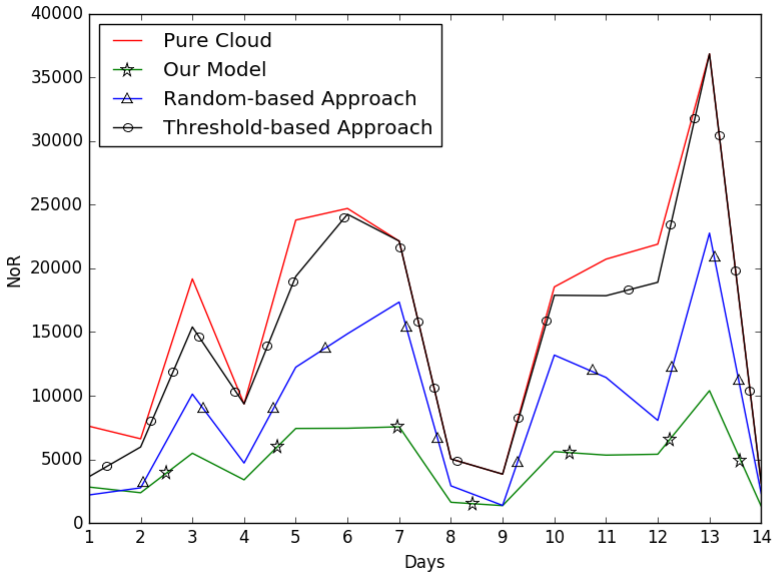


Figure 24: A comparison of our model to the Random-based [17][27] and Threshold-based [39] approaches showing the number of requests received by the cloud, all having a punishment value = 5 Hrs.

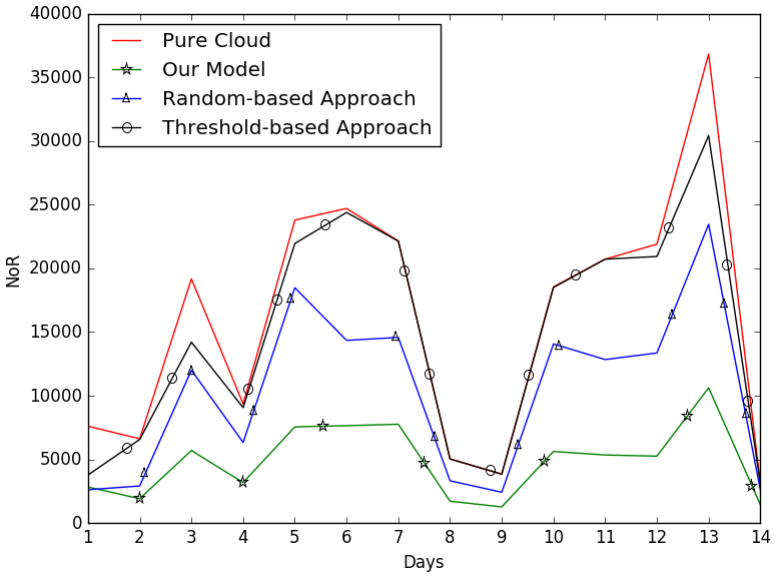


Figure 25: A comparison of our model to the Random-based [17][27] and Threshold-based [39] approaches showing the number of requests received by the cloud, all having a punishment value = 10 Hrs.

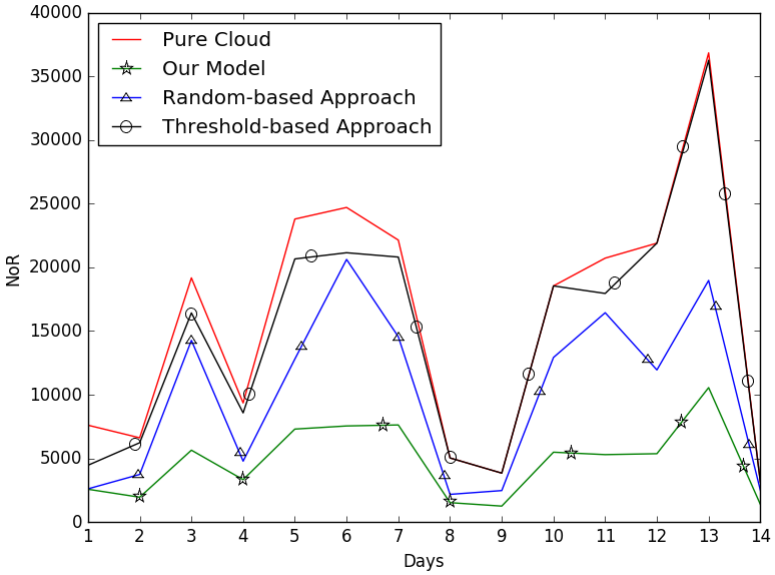


Figure 26: A comparison of our model to the Random-based [17][27] and Threshold-based [39] approaches showing the number of requests received by the cloud, all having a punishment value = 15 Hrs.

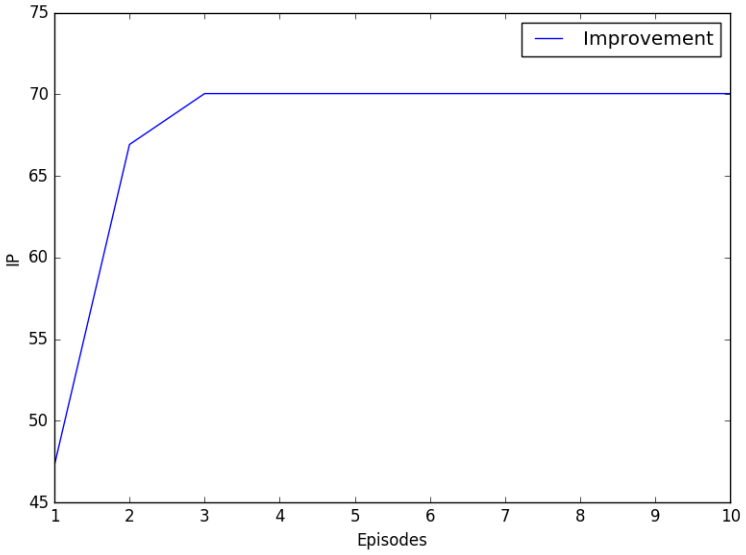


Figure 27: Accuracy convergence of our model with respect to the increasing number of training episodes.

the accurate predictions done by our model towards decreasing the number of requests processed by the cloud, regardless of the punishment value. As we can see in Figures 24, 25 and 26, our model achieved better results compared to the other two approaches. It was able to successfully predict to what location and at which time a fog must be created to decrease the processing load performed by

the cloud. In the three figures, at day = 1, the difference between our suggested model and the other two approaches is quite small. This is due to the lack of knowledge at the early stages of the process. As time progress, the gap between the curves starts to expand to show the important effect of the suggested model compared to the other two. As we increase the punishment value, which can be directly linked to the cost value of creating a fog, our model showed a wider gap between its curve and the other two. This is simply due to restricting randomness by punishing it, which helps in decreasing the cost in general. For example, at day = 3 in Figure 24 which represents a total of around 20000 incoming requests to the cloud, the value of the number of requests processed by the cloud becomes close to 15000 request while using the threshold-based approach, around 10000 request while using the random-based approach, and around 5000 request while using our model. If we look at the same day in Figure 26 where the punishment value is increased, we can see that the value of processed requests by the cloud is the same (close to 5000 request) when using our model, while the number of processed requests using the threshold-based and random-based approaches expanded to reach around 16000 and 15000 request respectively. One might ask about the reason behind always having better results for the random-based approach compared to threshold-based one. Because of the fluctuating behaviour of the requests, the threshold-based approach will have bad results because it resides entirely on the predefined threshold to decide. Since we have a punishment value while applying this approach, the fog will stay in the same location for the predefined punishment value which forbids it from serving better locations. On the other hand, the random-based randomly chooses locations regardless of the current value. This will allow it to have a bigger probable number of requests to serve if it performs a lucky guess. If we take another scenario, day = 6 for example, where the pure total number of incoming requests is 25000, we can see that in Figure 24, the number of requests processed by the cloud after applying the threshold-based approach, the random-based approach and our model becomes

around 24000, 15000 and 7000 respectively. Observing the same case in Figure 26, we can see that the number of requests processed by the cloud increased to reach 20000 request while using the random-based approach, decreased to reach 20000 requests while using threshold-based approach, and maintained same result while using our approach. The main reason behind the increase and decrease in the number of requests while using the random-based and threshold-based approaches at that day is due to the fact that the number of requesting locations became bigger (increase while applying random-based approach) while having few specific dominating locations sending the majority of the requests (decrease while applying threshold-based approach). Between day 8 and 9, and because of the small number of incoming requests compared to other days, the gap between the curves decreased. The random-based approach showed relatively close results to our model, which is strictly related to the small number of incoming requests and locations that were requesting at that specific date and time. Since the random-based approach randomly chooses a location and sets a fog near it, and since there is a small number of locations to choose from, both our model and the random-based approach outputted close results.

To further show the effectiveness of our approach, we created different episodes where the model loops over the same input list while keeping the data in its Q-table for several iterations. Figure 27 shows the improvement percentage of the ML model w.r.t decreasing the processing load performed by the cloud. The values of the graph are calculated by dividing the total number of requests processed by the fogs per episode, over the total number of requests processed by the cloud in that same episode, all multiplied by 100. At episode = 1, the best achieved result was having around 47 percent of the requests being processed by the fogs. This percentage increases to reach 70 percent of the requests being processed by fogs after looping over the same set of for three times. The Q-table converges at the third episode and achieves the best results possible using the 20 given fogs over the 65 different locations. Reaching a 100 percent means that all the

requests are being processed by fogs. This can only be achieved if the number of available fogs is sufficient for covering all locations. In our case, having 20 fogs to cover 65 location can never yield the 100 percent result. This experiment proves that if the algorithm is fed with enough to learn the behaviour of the requests, it will be able to maximize the benefits and best achieve the desired conditions.

The on-demand fog creation model and the methodologies suggested in this chapter satisfied the main objective which is decreasing the number of requests reaching the cloud by giving the most optimal schedule of fogs over locations. The ability of learning the behaviour of users regarding service requests using machine learning, proves that this model outperform the other approaches suggested in the literature.

Chapter 5

Conclusion

In conclusion, we have proposed new and intelligent approaches for solving two of the most important problems when it comes to cloud/fog/user environment. The proposed model for solving the fog localization problem relied on a hybrid technique made up of two famous machine learning concepts which are time-series and reinforcement. After introducing the ability of learning the behaviour of the end users, the model was able to predict the most adequate distribution of fogs over locations in a way that minimizes the load reaching the cloud. Solving such a problem by providing the best distribution of fogs comes with a myriad of advantages such as decreasing the network congestion, increasing the quality of service, decreasing delays, decreasing cost and much more. The promising results of our approach proves that introducing machine learning in such fields is beneficial and important compared to other methods and models that are currently present in literature. The story for this environment could not have been fully completed without the introduction of another model (an enhancement) which allows the cloud to distribute not only the fogs over locations, but also the containers/services over fog nodes. The enhanced GA approach which also uses machine learning is based on a clustering technique which clusters both hosts and services according to their resources in a way that allows us to form an intelligent linkage between the requirements of the services/containers and the providers which are the hosts. We proved via our experiments that the enhanced

version of the GA was able to excel in generating good quality solutions in a fast and more feasible way compared to the MA that is proposed in literature. Once more, with another problem, we were able to come up with a solution that uses machine learning to generate intelligent solutions. We were able to deliver significant improvement regarding both problems and we proved in this work that we can always make use of machine learning to solve problems that are usually hard to solve using today's methods and models.

Bibliography

[1] Nasa dataset - two months of http logs from a busy www server.

[2] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi. Dynamic task offloading and scheduling for low-latency iot services in multi-access edge computing. *IEEE Journal on Selected Areas in Communications*, 37(3):668–682, 2019.

[3] A. M. Alberti and D. Singh. Internet of things: perspectives, challenges and opportunities. In *International Workshop on Telecommunications (IWT 2013)*, 2013.

[4] K. Alsabti, S. Ranka, and V. Singh. An efficient k-means clustering algorithm. 1997.

[5] R. Arora, A. Parashar, and C. C. I. Transforming. Secure user data in cloud computing using encryption algorithms. *International journal of engineering research and applications*, 3(4):1922–1926, 2013.

[6] C. Bandt and B. Pompe. Permutation entropy: a natural complexity measure for time series. *Physical review letters*, 88(17):174102, 2002.

[7] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012.

[8] P. J. Brockwell, R. A. Davis, and M. V. Calder. *Introduction to time series and forecasting*, volume 2. Springer, 2002.

[9] S. Dhakate and A. Godbole. Distributed cloud monitoring using docker as next generation container virtualization technology. In *India Conference (INDICON)*. IEEE, 2015.

[10] E. Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of heuristics*, 2(1):5–30, 1996.

[11] P. Farhat, H. Sami, and A. Mourad. Reinforcement r-learning model for time scheduling of on-demand fog placement. *The Journal of Supercomputing*, pages 1–23, 2019.

[12] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and linux containers. In *2015 IEEE International Symposium Performance On Analysis of Systems and Software (ISPASS)*. IEEE, 2015.

[13] X. Gao. Deep reinforcement learning for time series: playing idealized trading games. *arXiv preprint arXiv:1803.03916*, 2018.

[14] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29, 2013.

[15] Z. Hao, E. Novak, S. Yi, and Q. Li. Challenges and software architecture for fog computing. *IEEE Internet Computing*, 21(2):44–53, 2017.

[16] H.-J. Hong, P.-H. Tsai, and C.-H. Hsu. Dynamic module deployment in a fog computing platform. In *2016 18th Asia-Pacific on Network Operations and Management Symposium (APNOMS)*, pages 1–6. IEEE, 2016.

[17] S. Hoque, M. S. de Brito, A. Willner, O. Keil, and T. Magedanz. Towards container orchestration in fog computing infrastructures. In *Proceeddings of the 2017 IEEE 41st Annual on Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 294–299. IEEE, 2017.

[18] N. Kherraf, H. A. Alameddine, S. Sharafeddine, C. Assi, and A. Ghrayeb. Optimized provisioning of edge computing resources with heterogeneous workload in iot networks. *IEEE Transactions on Network and Service Management*, 2019.

[19] N. Kratzke. About microservices, containers and their underestimated impact on network performance. *arXiv preprint arXiv:1710.04049*, 2017.

[20] S. Kumar, R. K. Bhatia, and R. Kumar. K-means clustering of use-cases using mdl. In *International Conference on Computing and Communication Systems*, pages 57–67. Springer, 2011.

[21] R. Mahmud, R. Kotagiri, and R. Buyya. Fog computing: A taxonomy, survey and future directions. In *Internet of everything*, pages 103–130. Springer, 2018.

[22] W. Marrouche and H. M. Harmanani. Heuristic approaches for the open-shop scheduling problem. In *Information Technology-New Generations*, pages 691–699. Springer, 2018.

[23] K. K. Patel, S. M. Patel, et al. Internet of things-iot: definition, characteristics, architecture, enabling technologies, application & future challenges. *International journal of engineering science and computing*, 6(5), 2016.

[24] X.-Q. Pham and E.-N. Huh. Towards task scheduling in a cloud-fog computing system. In *2016 18th Asia-Pacific network operations and management symposium (APNOMS)*, pages 1–4. IEEE, 2016.

[25] 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.

[26] G. Sayfan. *Mastering Kubernetes*. Packt Publishing Ltd, 2017.

[27] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner. Optimized iot service placement in the fog. *Service Oriented Computing and Applications*, 11(4):427–443, 2017.

[28] E. M. Tordera, X. Masip-Bruin, J. Garcia-Alminana, A. Jukan, G.-J. Ren, J. Zhu, and J. Farré. What is a fog node a tutorial on current concepts towards a common definition. *arXiv preprint arXiv:1611.09193*, 2016.

[29] H. Tout, C. Talhi, N. Kara, and A. Mourad. Smart mobile computation offloading: Centralized selective and multi-objective approach. *Expert Systems with Applications*, 80:1–13, 2017.

[30] N. B. Truong, G. M. Lee, and Y. Ghamri-Doudane. Software defined networking-based vehicular adhoc network with fog computing. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 1202–1207. IEEE, 2015.

[31] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[32] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl, et al. Constrained k-means clustering with background knowledge. In *Icml*, volume 1, pages 577–584, 2001.

[33] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad. Resource-aware detection and defense system against multi-type attacks in the cloud: Repeated bayesian stackelberg game. *IEEE Transactions on Dependable and Secure Computing*, 2019.

[34] O. A. Wahab, N. Kara, C. Edstrom, and Y. Lemieux. Maple: A machine learning approach for efficient placement and adjustment of virtual network functions. *Journal of Network and Computer Applications*, 2019.

[35] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[36] D. Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[37] S. Yaqoob, A. Ullah, M. Akbar, M. Imran, and M. Shoaib. Congestion avoidance through fog computing in internet of vehicles. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–15, 2019.

[38] S. Yi, C. Li, and Q. Li. A survey of fog computing: concepts, applications and issues. In *Proceedings of the 2015 workshop on mobile big data*, pages 37–42. ACM, 2015.

[39] A. Yousefpour, G. Ishigaki, and J. P. Jue. Fog computing: Towards minimizing delay in the internet of things. In *2017 IEEE International Conference on Edge Computing (EDGE)*, pages 17–24. IEEE, 2017.

[40] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu. Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system. *IEEE Transactions on Computers*, 65(12):3702–3712, 2016.