The Journal of Supercomputing (2020) 76:388–410
https://doi.org/10.1007/s11227-019-03032-z

Reinforcement R-learning model for time scheduling of on-demand fog placement

Peter Farhat[1] · Hani Sami[1] · Azzam Mourad[1]

Published online: 22 October 2019

Abstract

On the fly deployment of fog nodes near users provides the flexibility of pushing services anywhere and whenever needed. Nevertheless, taking a real-life scenario, the cloud might limit the number of fogs to place for minimizing the complexity of monitoring a large number of fogs and cost for volunteers that do not offer their resources for free. This implies choosing the right time and best volunteer to create a fog which the cloud can benefit from is essential. This choice is subject to study the demand of a particular location for services in order to maximize the resources utilization of these fogs. A simple algorithm will not be able to explore randomly changing users' demands. Therefore, there is a need for an intelligent model capable of scheduling fog placement based on the user's requests. In this paper, we propose a Fog Scheduling Decision model based on reinforcement R-learning, which focuses on studying the behavior of service requesters and produces a suitable fog placement schedule based on the concept of average reward. Our model aims to decrease the cloud's load by utilizing the maximum available fogs resources over different locations. An implementation of our proposed R-learning model is provided in the paper, followed by a series of experiments on a real dataset to prove its efficiency in utilizing fog resources and minimizing the cloud's load. We also demonstrate the ability of our model to improve over time by adapting the new demand of users. Experiments comparing the decisions of our model with two other potential fog placement approaches used for task/service scheduling (threshold based and random based) show that the number of processed requests performed by the cloud decreases from 100 to 30% with a limited number of fogs to push. These results demonstrate that our proposed Fog Scheduling Decision model plays a crucial role in the placement of the on-demand fog to the right location at the right time while taking into account the user's needs.

Keywords IoT · On-demand fog · Fog scheduling decision · Average reward · R-learning · Q-learning · Reinforcement learning · Cloud computing

✉ Azzam Mourad
Azzam.mourad@lau.edu.lb

Extended author information available on the last page of the article

1 Introduction

The increasing number of IoT (Internet of Things) devices [1] having limited resources led to the presence of fogs that are deployed at the edge of the network by extending the cloud solutions [2, 3]. Fog nodes [4] tend to assist such constraint devices that require fast processing and low networking delays to maintain an acceptable level of quality of service (QoS) for the cloud applications [5]. The concept of fog computing can be illustrated as a software running on a device that collects, processes, and sends data on behalf of the IoT devices to the cloud. The main advantage of deploying fog nodes is performing some processing on behalf of the cloud to minimize the resources load on it and achieve a lower response time while avoiding the propagation delays to/from the user. This strategy helps to avoid network congestion and save energy on the requesting devices. In our previous work [6], we utilized the volunteering resources present everywhere to become fog devices and enhance the fog availability. The flexibility of initializing any type of device as fogs was made possible through our on-demand fog formation framework. Lightweight services are pushed as containerized micro-services and monitored using the Kubeadm orchestration tool that divides volunteers into clusters [7, 8]. In our framework, we let the cloud take the time scheduling decision and assumed that optimal decision was always taken by default. To the best of our knowledge, none of the related literature has implemented a decision model that considers user's historical behavior of requesting a service to decide on the proper time an location to place that service on fog devices.

Pushing on-demand fogs to volunteering devices everywhere has its disadvantages also. Here we mention the potential high complexity of monitoring all fog placements [9], which the cloud is responsible for. Moreover, having to push services everywhere can be considered as a security threat for the service provider, especially when services get controlled by volunteers [10]. In addition, volunteers will, at some point, ask for a reward for their offered services, which the cloud should pay [11]. All of these factors lead to the need for minimizing the number of fog placement. Therefore, these are considered as the main motivations behind limiting the number of fog placement by the cloud. This now raises another challenge on the cloud, which is making fog placement decisions at the time and place that best maximizes its profit by maximizing the utilization of fog resources. Simple approaches are not enough for the cloud to make good scheduling decisions for cloud placement. However, there should be a model that studies users' demands to decide on the best time and place to schedule the fog. Pushing fogs to places having high demands of services can significantly improve the decision efficiency by maximizing the fog resources utilization and therefore minimizing the load on the cloud by having to deal with fewer requests. An example of possible approaches for fog scheduling is the threshold-based and random-based approaches which we prove in the paper that they are not applicable in real life.

In this paper, we address the aforementioned problem by proposing a Fog Scheduling Decision model based on a reinforcement learning technique called

R-learning. R-learning has a similar concept to the Q-learning technique. They both use Q-tables, states, actions, reward, and loss. However, the main difference is that R-learning uses the concept of average reward and not the discounted reward [12]. In other words, our problem does not converge to a static end goal, but rather a general goal of minimizing the cloud load. This is done by maximizing the total average reward of transitioning between all states and taking actions. R-learning is an area of reinforcement learning that aims to maximize the total reward by taking a certain action on a given state and performing punishment whenever the model fails to meet the expectation [12, 13]. In this context, our proposed approach predicts the time and location of the needed fogs that mostly minimize the number of requests processed by the cloud while serving the biggest number of requests generated by users or IoT devices and therefore maximize the usage of volunteering resources. The predictions of this algorithm help in increasing the QoS as well as maintaining a low pressure on the cloud resources. This is done through studying user's behavior of making requests to services hosted on the cloud. Users' requests can be tracked using the server's logs. A real dataset is used from the logs of the NASA server [14], and experiments are conducted on the implemented scheduling model to prove its efficiency and enhancements achieved over time by taking near-optimal service scheduling decisions. We also compare our proposed model to the decisions taken by the threshold- and random-based approaches to show the achieved improvement.

The rest of this paper is organized as follows: In Sect. 2, we present some background information and study the current work surrounding the use of the time scheduling model and the effect of its absence. In Sect. 3, we introduce an overview of the architecture and methodology used where our proposed model plays the main role. In Sect. 4, we propose the Fog Scheduling Decision model and explain the mathematical formulas and R-learning algorithm behind it. Section 5 is dedicated to analyzing the results generated by our experiments. Finally, we conclude the paper and present future directions in Sect. 6.

2 Related work

Trying to schedule task placement on the spot when the user requests to have one nearby or based on standard criteria such as threshold and ad hoc decision-making is not relevant when a lot of services need to be pushed on a specific number of volunteers. To the best of our knowledge, we are the first to tackle the problem of deciding on the proper time and place to schedule a task or a service placement in the form of containers on a defined number of available fogs. Therefore, in this section, we discuss the different work in the literature that misses tackling this decision model.

We first go over some of the surveys discussing current fog's open problems and challenges. We then summarize existing work that considers tasks or services scheduling on available resources, and others that use containers to push services on the fly.

2.1 Fog computing existing challenges

In recent surveys [11, 15, 16], authors have highlighted the need for a proper model that schedules tasks and services in the form of containers on available fogs' resources, while emphasizing on its crucial effect on any further studies in this area. This scheduling is divided into two parts: The first is the proper decision on the time and place where the service should be pushed, which is our main area of interest in this paper. Second is the way of distributing the services on different heterogeneous available resources, which is mapped to the container placement problem in our previous work [6]. Various approaches in the literature have also considered solving the problem of placing tasks on resources [17–19], where heuristics was proved to take an accurate selection and placement decisions. However, predicting the right time and best place to assign the service to remains an open problem because of the non-deterministic user behavior and random requests of service assignments.

2.2 Task scheduling on available fog resources

Several approaches are meant to schedule task offloading on fog devices in two ways. From user to fog, so the user makes the offloading decision [20, 21], and from fog to cloud in case heavy task cannot be processed by fogs [22, 23]. However, none of them considered scheduling services from cloud to fogs based on user's demands in an automated way. The work in Zeng et al. [24] considers the scheduling problem of tasks on available fogs. Task processing can be done on two fog layers, either on an embedded device with limited resources near the client or on a server with high computation power. A fog server is capable of handling multiple tasks at the same time in contrast to the embedded devices. Besides, reaching the server to execute required assignments by the embedded devices can be subjected to network latency. Therefore, balancing the load on these fogs and scheduling the tasks on the proper device are highly relevant. To solve such a scheduling issue, they proposed a heuristics solution that aims to minimize the execution time of jobs on available resources. Results are promising in terms of reducing such time. On the other hand, this work did not study the behavior of users requesting the task concerning time and resources usage of the embedded device or server. This is important because a particular task scheduled on a server at time $t = 1$s can better utilize these resources for a small task scheduled at $t = 0$s. In Pham and Huh [23], users' requests are always received by a broker that manages available resources on the cloud and fogs, monitors the communication costs between them, and decides on the optimal task scheduling. Heuristics solution is proposed to schedule tasks between fogs and available cloud resources based on the task priority and resources available on fog nodes. Two major limitations of [24] and [23] are discussed as follows: First, both the above-mentioned studies miss considering the main factor of scheduling tasks depending on users' behavior of a particular location to be able to predict the fogs load. Second, they use heuristics to solve the scheduling problem while looking only at the current state in terms of resources requirements. Therefore, their solutions are not able to improve over time. These factors are considered the main motivation behind our work in this paper. A learning model that is taught by experience from a changing dataset is capable of scheduling adequate time and place for task offloading or fog placement as illustrated later in our experiments. Kherraf et al. [25] and Alameddine et al. [26] are trying to solve the similar problem of resource provisioning on edge/fog devices by dividing it into three subproblems: (1) Properly distribute the workload on the edge server, (2) decide on the proper placement of applications that needs to be pushed to process IoT generated, and (3) decide on the proper number of instances and locations to place these edge servers on. The authors formulated these problems mathematically using a mixed integer program and proposed a decomposition approach to solve them. On the other hand, we also point on the limitation of deciding on the proper time to schedule such placement, service deployment, and resource balancing decisions, which this work did not address. Authors in [27] try to minimize the time delay experienced by IoT devices whenever a fog is overloaded. Once an IoT device requests a service, the receiving fog estimates the waiting time for the task to be processed, which depends on the fog load. After setting a delay threshold, the fog decides whether to offload the task to a neighboring fog with less load to perform the processing. Setting a delay threshold is similar to placing a limit on the number of requests received, which can lead to offload the task or push service. Using a threshold is not optimal because the behavior of the requests is subject to change over time. In our experiments, we refer to this strategy as threshold based. We implement the threshold strategy described in Sect. 4.1 and prove, by experimentation, the advantage of R-learning model compared to it.

In [28], the cloud waits for the user to request a service placement. Because user behavior is random and capable of sending a placement request anytime, we refer to this strategy as random scheduling. We implement randomness described in Sect. 4.2 and prove through experiments how our approach outperforms the ad hoc/random strategy of pushing services.

2.3 Fog computing solutions using containers

Different procedures use containerization technology to enable on-demand fog placement. Therefore, it is worth mentioning the main work that considered the use of containerization technology in the fog computing context and the effect of not adapting a solution for proper fog scheduling. Authors in [29] were able to prove the ability to push services on preselected fogs using containerization technology. An orchestration layer is used to manage the running services. This work contribution was limited to proposing a solution to place services on fogs, without considering the possibility that the number of services or tasks that should be executed might require more resources, and not all of them can be scheduled for placement. Similarly, a model was proposed by the authors in [30], where the dynamic deployment of services on helper nodes(fogs) of the main server using Docker is possible. So it is feasible to remove, add, stop, and run any service on a physically known fog anytime. The Kubernetes orchestrator was running on a server. Their approach was first to gather users' requests on that server. Second, although fogs are not near users, the proposed model distributes requests on fogs after pushing the needed services. This means that fogs are only doing some processing instead of the server. Furthermore, networking delays are not avoided. However, they are getting faster processing time because of distributing the tasks from one server to many nodes. The main limitation of these two approaches is the missing factor of studying the proper assignment of tasks or services to available fogs. In case only a limited number of volunteering can be used, prioritizing services for scheduling can affect the framework efficiency. Therefore, the proper historical and behavioral study of users' services requests, resources availability to use, and adequate time and place to assign the service are necessary for any approach serving placement of services on the fly.

3 Architecture and methodology

In our previous work [6], we were able to propose an on-demand framework capable of creating fog nodes on volunteering devices anywhere and anytime. Although resource availability is not enough for building fog devices, services shall be migrated in lightweight fashion with the least costs. Therefore, we used the containerization technology to migrate small-sized micro-services and be adapted to run on any base operating system, i.e., any volunteering devices. Even though volunteering resources have limited resources capacity to host services, we suggested building Kubeadm clusters of available volunteers to augment computation power and enhance the distribution of micro-services. This being said, our framework is still missing a realistic fog scheduling model to be fully autonomous. Thereafter, this section is dedicated to describing the building blocks and functionalities of our fog scheduling model, preceded by a description of our on-demand fog formation framework that led us to the importance of building it. Studying user's behavior while requesting services is hard to do with a simple machine learning model. Therefore, we used reinforcement learning to learn user behavior by experience and adapt as services requirements per location change. In this section, we discuss the overall architecture combined with our learning decision model shown in Fig. 1 by dividing it into three main layers as follows: cloud, Kubeadm fog clusters, and users.

- 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 for making decisions concerning fog scheduling and localization. The agent is based on an R-learning model detailed in the next section. The primary 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 areas 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. By doing so, and by using techniques derived from R-learning methodologies, the agent will be able with time to make reasonable 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 Fig. 2. These requests are made up of an IP address (which is used

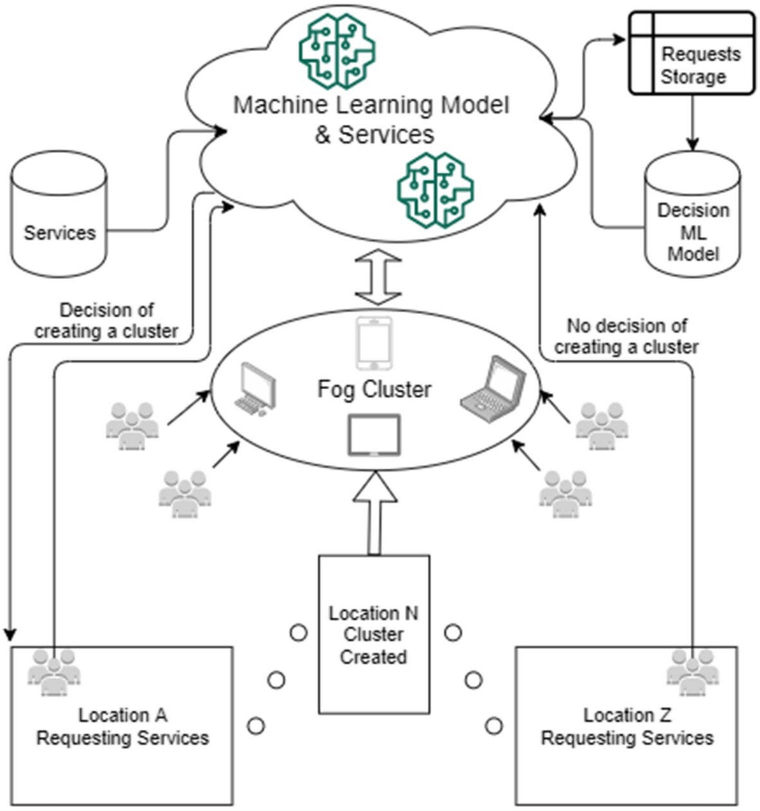
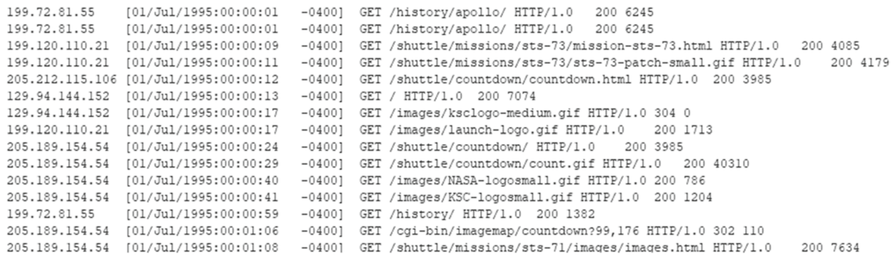

Machine Learning Model & Services
Requests Storage
Services
Decision ML Model
Decision of creating a cluster
No decision of creating a cluster
Fog Cluster
Location N Cluster Created
Location A Requesting Services
Location Z Requesting Services

Fig. 1 Overall model architecture



```
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
```

Fig. 2 A sample of the requests captured by the NASA server [14]

to determine the location of the user), date and time, and requested service. We will elaborate more on the request's format and usage later in this paper. To provide the reader with the general idea of how things work, a brief example of the overall process is presented next. Suppose the agent learns with the time that a group of users at location **L** are going to be requesting a lot of services at 4:00 p.m., 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 Fig. 1 constitutes the Kubeadm fog clusters. In our previous work [6], we have suggested a new concept of creating on-demand Kubeadm clusters, which aims to serve users. For a number of reasons, an architecture that combines both containerization and micro-service technologies was 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 (fogs), which are chosen by the master node, are responsible for hosting the services and serving users. In this approach, the Kubeadm cluster is created whenever a group of volunteers can collaboratively form the cluster. As a relation to what we have mentioned above, the orchestrator pushes the demanded services to targeted Kubeadm cluster in a specific location once the cloud decides to push a fog there. In our previous work, we assumed that the cloud always takes the best decision on the time and place scheduling of fogs by default. On the contrary, this work serves as an extension to propose such a scheduling model to result in a more efficient, automated, and realistic framework.
- The third and final level contains the users. Any newly connected device trying to use a service shall be initially routed directly to that cloud. If any existing Kubeadm clusters are near that device, the cloud routes that device to the nearest cluster to obtain the service. On the other hand, if no adjacent clusters are available, the cloud serves the user while recording the incoming requests. Users' requests are the source of learning for our R-learning agent, which is taught by experience to represent the need for a group of users or locations. The favorable decision of creating a fog near users results in better user experience by claiming less networking delays, faster computing via dividing load on volunteering fogs, and distributing the computation through micro-services and minimizing the load on the cloud for better performance.

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. Having the database on the cloud level is mainly because the cloud is the main server for IoT devices that handle their requests. The list of services is also placed on the cloud in order to have a centralized repository of services and better control their placements. The agent is the component that decides upon services placement by studying the behavior of users from the data present in the database. Another motivation for placing the agent on the cloud is to monitor its placement decisions in order to update its model and keeps learning from all user's requests. The link between the second layer containing the fog cluster and the agent is for calculating the reward or loss resulted from the decision taken. The model functionality is described in the next section. The second layer comprises the Kubeadm clusters, which use the containerization technology for services deployment. The primary motivation behind this layer is thoroughly discussed in our previous work [6].

To satisfy this requirement, we propose in this paper the **Fog Scheduling Decision** based on the reinforcement R-learning model, which is presented in the next section. Our proposed model guarantees a better user experience and wise Fog Scheduling Decision through adequately employing volunteering resources at the right time and place.

4 Fog Scheduling Decision model

In the previously used on-demand fog formation architecture, we assumed that the decision to create fog is taken by default. Considering the threshold-based approach, if the number of requests exceeded a predefined 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. For this reason, we define in this section our fog scheduling problem to be formulated and solved using a reinforcement R-learning approach. Our approach is model-free and learns by experience.

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. The reason behind setting a number of fogs to be pushed is to take into account any costs paid by the cloud to volunteers, reduce the complexity of managing lot of fogs, and provide optimal resource management. Therefore, the resources usage on the fog devices should be maximized, or in other words, the user must be in need for a fog nearby since being able to infinitely push fogs to volunteers in all locations is most likely to not occur in real life.

4.1 R-learning problem definition and challenges

4.1.1 Problem definition

Given the historical behavior of users represented as the number of requests generated from locations L at the specific time of given days, and a number of fogs M to be pushed, the cloud should use these data to make appropriate decisions to either push a service or not to every location in L based on the hours of the day. For this paper, **our model is environment-free** and does not study what services particularly to push, but rather either a fog shall be created or not only. The creation of fog should result in a maximization of fog resource utilization, therefore service more users, which implies higher QoS and satisfaction for users. Deciding what locations are really in need of fog presence also increases the cloud performance by minimizing the load and at the same time better costs utilization from cloud to volunteers, if any. The reward function that should be maximized is discussed in Sect. 4.3.1 Eq. 1.

As previously mentioned, in order to provide a feasible solution to study the random behavior of users, represented as locations need in our case, we decided to create an R-learning algorithm that learns by the experience each day, wins rewards when issuing a good decision, and be punished or looses rewards by any wrong decisions. The main challenges of formulating our problem as an R-learning are mentioned in the next subsection.

4.1.2 R-learning background and challenges

In this paper, we are formulating our model as model-free to be solved using Q-learning average reward, called R-learning. The reason behind choosing a model-free agent is that we are not able to predict the dynamics of the environment, or we cannot set a probability transition matrix between the states. Therefore, modeling the environment will make it complex.

A reinforcement Q-learning problem is defined as the Q-table, which contains states and actions (2D matrix), reward and punishment formulas, updating Q-function, initial state, and an end goal. The model first initializes the Q-values of a Q-table either to zero or to random negative numbers (for example, between -2 and 0). The exploration phase of the model starts by taking random actions at the beginning. At a given state, the action having the highest Q-value is considered because it is expected to maximize the reward and reach the end goal. By taking action for a given state, the model should be able to decide on the next state to go to. When deciding on the next state to go, the model updates the Q-value of the current state based on the action taken. The algorithm then keeps on updating the Q-table until reaching the end goal, for example having a Q-value $= 0$.

The two main challenges in our reinforcement Q-learning problem definition are: to first decide on the end goal and second define the states and actions that make up the Q-table.

4.1.3 Average reward-based R-learning

In a standard Q-learning problem, the learning or updates of the Q-table terminates whenever the algorithm reaches the end goal. An instance of such a case is Mountain Car Problem [31]. In this problem, the car starts at position (0, 0) with speed equal to zero. The states are defined as the car coordinate in (x, y) plane and velocity range. The actions are defined as the direction of movement (left, neutral, right). The end goal is to guide the car to reach the top of the mountain, resulting in a Q-value equal to zero. Therefore, this problem uses a “discounted reward,” where an immediate reward is learned after each state transition. In contrast, our fog decision problem does not have one end goal, but a way to maximize the reward by selecting the right volunteers and taking into account the random demands of users in locations. Therefore, there is no guaranteed convergence for our model. In other words, our Q-table keeps on changing whenever new demands appear from locations. In this situation, we decided that the end goal would be to maximize the average reward for the iterations of a given day. Therefore, we used reinforcement Q-learning based on **Average Reward** called

R-learning. In R-learning, the average reward that should be maximized is a result of rewards earned during the entire transition between states [32]. The first main challenge is on deciding upon the use of Average Reward-based R-learning.

4.1.4 Q-table definition

Each Q-table represents the decisions of one day. A state is represented by a location and hour on a specific day. The first M states in our Q-table represent all locations during the first hour of the day. Therefore, the number of states in our Q-table is $M \times 24$, which are the number of locations times the 24 hours of the day. The actions are either to push or not to push. Therefore, the dimensions of our Q-table matrix are $(M \times 24) \times 2$. In addition, we are using the transition strategy between states to be sequential on all possibilities of hour and location. In other words, the algorithm transitions between states sequentially and from one hour to another to explore all states' possibilities. Our Q-table is represented as follows:

$$\begin{bmatrix} & \text{Push} & \text{DoNotPush} \\ \text{Loc } 1, 0- & - & \\ \text{Loc } 2, 0- & - & \\ \text{Loc } 3, 0- & - & \\ \cdots & \cdots & \cdots \\ \text{Loc } 1, 12- & - & \\ \text{Loc } 2, 12- & - & \\ \text{Loc } 3, 12- & - & \\ \cdots & \cdots & \cdots \\ \text{Loc } M-2, 23- & - & \\ \text{Loc } M-1, 23- & - & \\ \text{Loc } M, 23- & - & \end{bmatrix}$$

4.2 Inputs–data feeding

4.2.1 Data description

It is sufficient to have historical records of the requests initiated by users, including their locations and time of each request for a given day. The location can be extracted from an IP address. For instance, we consider the first octet of an IP address to identify a location. This information can be usually found in any server logs. As previously mentioned, this work does not make a decision upon choosing what services should be pushed to specific locations, but rather one decision to specify the places with the highest need of fogs. Therefore, the exact service name requested by a user is not required for our proposed model to work. A sample of how a request looks like is shown in Fig. 2.

4.2.2 Data loading

In order to better represent the data, we extract the information needed to a 3D matrix. We call this matrix the *learningData*. It is formatted as follows:

- *learningData*[i] represents a dictionary of states of the form (hour, location) where requests on the ith day come from.
- *learningData*[i][j] represents the list of requests sent at state j on day i.
- *learningData*[i][j][k] represents the time when request k is sent at state j on day i.

An example of the learningData output is shown in Sect. 5.3.

4.2.3 Learning by experience

In reinforcement learning, there are no data for the model to start training on. The Q-table is at the beginning either initialized to zero or to random values. Therefore, the model starts the exploration phase by taking actions randomly. The model is rewarded for good decisions and punished for bad ones. Once the first day finishes, the model should have gotten the first insight from the first-day data by updating its Q-values. Once the second day comes, the model takes a decision at each state either randomly (non-greedy approach) or by selecting the action having the highest Q-value (greedy approach). The model then keeps issuing decisions each day and learns from the action taken by propagating the reward or loss to the Q-table using a Q-function described later. As long as new data are coming, the model keeps on improving by updating its Q-table. The model is not expected to converge because it should adapt to any unexpected demand from any location.

4.3 Learning phase

Our R-learning agent takes actions based on values present in the Q-table. Our updating formula should result in a positive reward once resulted in a good decision, and a negative one otherwise. Therefore, in this section, we define the reward and punishment function calculated after taking a push or do not push action. Once the reward is calculated, this value should reflect in the Q-value of the current state and be propagated in the Q-table using a Q-function, also defined in this section. We then present the full R-learning algorithm based on maximizing the average reward. Noting that all algorithms and equations, except Eqs. 2 and 3, are our own contribution.

4.3.1 Reward and punishment

Three components are responsible for building the reward function to represent better the goodness of the action taken. These components are: The number of requests received from the current location and hour (denoted as NoR), the total number of requests received from all locations during the current hour (denoted as TNoR), and finally the fog idle time which represents the time for which the fog did not receive any request from the covered location at a given hour (denoted as FogIdleTime). In the sequel, we show the algorithms (Algorithms 1–4) used for each component calculation, as well as the end reward function and the motivation behind it. The code in this paper is written in Python programming language.

Algorithm 1 *getNoR:* Get number of requests per state - NoR

```
    Input: hour, location & day
    Result: Number of requests
1: procedure NoR Retrieval
2:     NoR = Retrieve the number of requests from the learningData matrix using length(learningData[day][hour,location])
3:     Return NoR
```

Algorithm 2 *getTNoR:* Get number of requests per hour - TNoR

```
    Input: hour, Locations & day
    Result: Total number of requests per hour
1: procedure TNoR Retrieval
2:     $TNoR = 0$
3:     for each $location \in \mathcal{L}ocations$ do
4:         $TNoR \leftarrow TNoR + getNoR(hour, location, learningData)$
5:     Return TNoR
```

Algorithm 3 *getFogIdleTime:* Get fog idle time in minutes per state - FogIdleTime

```
    Input: hour, location & day
    Result: Fog idle time
1: procedure FogIdleTime Retrieval
2:     RequestTimes = Define an empty set containing minutes of the requests issuing time using Set()
3:     RequestsInfo = Retrieve a list of times when each request was issued for a state using learningData[day][(hour,location)]
4:     for each $requestTime \in \mathcal{R}equestsInfo$ do
5:         Add requestTime to RequestTimes using RequestTimes.add(requestTime)
    fogIdleTime = 60 - length(RequestTimes)
6:     Return fogIdleTime
```

Algorithm 4 *reward:* Reward function calculation

```
Input: state & action
Result: reward
1: procedure REWARD CALCULATION
2:     hour = state[0], location = state[1], day = state[2]
3:     $NoR$ = getNoR(hour,location,day)
4:     if $NoR = 0$ then
5:         Return -1
6:     $fogIdleTime$ = getFogIdleTime(hour,location,day)
7:     $TNoR$ = getTNoR(hour,location,day)
8:     $rewardResult = action * \frac{NoR - \frac{fogIdleTime}{NoR}}{TNoR}$
9:     Return $rewardResult$
```

The reward is calculated using the below equation:

$$\text{reward} = \text{action} \times \frac{\text{NoR} - \frac{\text{fogIdleTime}}{\text{NoR}}}{\text{TNoR}} \tag{1}$$

The action is equal to − 1 when the decision taken by our R-learning agent is not to push and 1 otherwise. In case the decision is to push, and it turns out to be a good decision, the equation will result in a positive reward. A good decision means a high NoR during the current state and less fodIdleTime, which means a good distribution of requests over time. Therefore, the numerator will result in a high value and large reward. In case the push decision is not sufficient, the numerator will result in a negative value because the NoR will be small, and the fogIdleTime is large. This negative reward is considered as a punishment value for our agent which results in minimizing the Q-value for the taken action.

In contrast, a decision of not pushing, having action equal to − 1, results in a positive reward when the agent does well by not pushing, and a negative reward otherwise.

The fogIdleTime is divided by the NoR in order to minimize the effect of fogIdleTime on the rewarding result when the number of requests at the current state is large, even if the idle time is not very small. On the other hand, the effect of fogIdleTime is large when NoR is small.

The reward is divided by the TNoR to better represent a rate of the NoR for the current location and result in a reward between − 1 and 1.

4.3.2 Q-function or Q-factor update

After taking action a at state i, Q(i, a) is updated using the below Q-function presented in [12]:

$$\begin{aligned} Q(i,a) =& (1-\alpha^k) \times Q(i,a) \\ & + \alpha^k \left[r(i,a,j) - \text{averageReward}^k \times t(i,a,j) + \eta \max Q(j,b) \right] \end{aligned} \tag{2}$$

Algorithm 5 *QFactorUpdate:* Q-Function calculation

```
    Input: QTable, stateI, stateJ, day, action & averageReward
    Result: QFactor
1: procedure QFactor Update
2:     $QOptionStateJ$ = Get Q-values of the next state 'j'. Example: QTable[states.index(stateJ)]
3:     $QValueStateJ$ = max(QOptionStateJ)
4:     $qValueUpdate = ((1 - \alpha) * QTable[states.index(stateI)][action]) + \alpha * (reward(stateI, action) - averageReward + (n * QValueStateJ))$ //Apply q-function and return the new q-value
5:     Return $qValueUpdate$
```

In contrast to any Q-function, our model uses R-learning, which implies the use of average reward to update the Q-factors or Q-values. Algorithm 5 shows how the Q-function function is defined in Python (to be used in Algorithm 6).

In Eq. 2, α is a learning rate discussed later in the paper. k represents the number of iteration. r(i, a, j) is the reward resulted from moving to state i to j after taking action a. p^k represents the average reward on iteration k. t(i, a, j) is the transition time from state i to j by taking action a. In our case, t(i, a, j) is the transition time between states and is equal to 1 in our case. η is a scaling constant less than 1 but close to 1. In our model, we used $\eta = 0.99$. $\max Q(j, b)$ is the maximum Q-value at state j.

4.3.3 R-learning algorithm

Our R-learning algorithm, presented in Algorithm 6, is divided into four steps. Below we present a description of each step:

- Step 1 (Input): The first step comprises the initialization of variables and **Q-table**, and updating α and β. **ITERMAX** represents the maximum number of iterations for one day in the **learningData** matrix, and **k** is the current iteration number. **states** list contains all states possible in our Q-table. Each state in the below algorithm is represented as (hour, location). As previously discussed, the first M states are tuples of the first hour, each having a different location. The first M states can be represented as (0, locations). The Q-table is initialized to zero. α and β are two learning rates variable less than one and are functions of k. These variables should converge near zero as k increases or as the model learns more. We followed [32] to chose α^k and β^k to be $\frac{log(k)}{k}$ and $\frac{90}{100+k}$, respectively.
- Step 2 (Q-value update): In this step, we apply the **random probabilistic approach** to decide whether to try exploring, or selecting the action having the highest Q-value. The algorithm then updates the Q-value of the current state based on the action taken and Q-function calculation.

  Following the approach of [32], we define a probability, function of k, called **pk**, to be $\frac{G_1}{(G_2+k)}$ where G_1 and G_2 are two constants such as $G_1 < G_2$. In our algorithm, we chose $G_1 = 10$ and $G_2 = 20$. This motivation behind calculating this probability is to leave room for exploration in the algorithm. In our R-learning algorithm, we get a random value between zero and 1. We get pk as a result of the probability function for a given k. In case randomValue $\leq (1 - pk)$, our agent chooses the action based on a **greedy** approach, or in other words taking action having the highest Q-value. Otherwise, the agent tries to explore other possibilities by taking action having the smaller Q-value, or following the **non-greedy** approach. This is important for our algorithm to keep learning new patterns and explore new decisions as to the demand change in some locations. As the value of k increases, the probability value decreases, and therefore, the model tends to explore less.

Based on the taken action, − 1 for not push and 1 for push, the agent then extracts the highest Q-value of the next state j. After that, the new Q-value for the current state is computed by calling the Q-function of Eq. 2.

- Step 3 (Updating Average Reward): After updating the Q-Value and getting the reward, the average reward should be updated. If the chosen action was non-greedy (following the exploration), the agent goes to Step 4. Otherwise, the total, time, and average rewards are updated as follows: $\text{totalReward} = \text{totalReward} + r(i, a, j)$ and $\text{timeReward} = \text{timeReward} + 1$. The averageReward is then updated using Equation 3, which was presented by [12].

$$\text{averageReward}^{k+1} = (1 - \beta^k) \times \text{averageReward}^k + \beta^k \times \frac{\text{totalReward}}{\text{timeReward}} \tag{3}$$

- Step 4 (Check for Termination): The agent increments k by 1 and returns to step 1 whenever k < ITERMAX. Once k reaches ITERMAX, the agent starts learning on the next available day on the learningData matrix to repeat the process again with k = 0. When the agent finishes learning on all days, it checks for additional epochs to improve its learning. In case there are no more epochs to process, the agent finally goes to step 5. Our model keeps on getting new data every day, so it learns from its previous action to improve the model further. Therefore, on every new day, the agent reenters a new cycle of learning by repeating all the steps.
- Step 5 (Generate a Policy): Once the model finishes its learning cycle on all data present in learningData matrix, it builds its policy to make decisions on a coming day. The policy is built by selecting the action (push or not push) for every state based on its maximum Q-value. Because the number of fogs that the cloud can push might be limited, for instance, **M** fogs, the agent selects M push decision having the highest Q-values in order.

Algorithm 6 *R-Learning*

```
Input: Requests
1: procedure LEARNING
2:     $ITERMAX = 200$, $epochs = 10$, $\alpha = 0$, $\beta = 0$, $\eta = 0.99$, States, QTable, learningData
3:     for each $epoch \in \mathcal{R}ange(epochs)$ do
4:         for each $dayState \in \mathcal{R}ange(length(learningData))$ do
5:             k=1, totalReward=0, totalTime=0, averageReward=0
6:             while $k < ITERMAX$ do
7:                 update $\alpha$ and $\beta$
8:                 for each $state \in \mathcal{S}tates$ do
9:                     Apply random probabilistic approach to decide if greedy approach is followed or not
10:                    Update QValue of current $state$ based on action taken using the $QFactorUpdate$ function
11:                    if Greedy approach is followed Then:
12:                        totalReward = totalReward + reward of current state and action
13:                        totalTime = totalTime + 1
14:                        Update average reward using equation 3
                   Increment k
15:     Generate the policy based on Q-table
```

5 Experiments

In this section, we first discuss the objectives of conducting our experiments. We then define the threshold- and random-based approaches to be compared with the decisions of our proposed model. After that, we discuss the dataset used to perform our evaluations. Finally, the experimental setup is explained, followed by the two evaluations performed to prove our model efficiency and significance.

5.1 Objectives

There are two main objectives for these experiments, as listed below:

- The first is to prove the above claimed functionality of the ability of the model to learn by experience. This is shown by the improvement in the model achieved over time.
- The second objective is to show the significance of our model and the good placement decisions based on time and location. This can be proved by outperforming other techniques existing in the literature, such as the threshold- and random-based decision-making for fog placement. In the next section, we describe the way used to implement each of the threshold- and random-based approaches.

5.2 Implementation of existing work approaches

5.2.1 Threshold-based approach

In this approach, a predefined threshold is used. This threshold approach is based on the literature work of [27] who considered the time as a threshold; however, in this experiment, we consider the number of requests as a threshold. If the number of requests received from a specific location at a specific hour exceeds that threshold, one of the **M** fogs would be used to serve that location. It is important to note 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 agent might push to locations prior to the knowledge that other locations might generate more requests. In contrast, in case the threshold is high, the agent waits until the threshold is reached by certain locations. This can lead to missing to serve other locations, which can also be in need. On the other hand, our model is capable of learning a pattern from the user behavior by counting on the reward and loss calculations reflected in the Q-table of the given state. Our approach advantage is shown in the study of Sect. 5.6.

5.2.2 Random-based approach

This approach, as the name states, follows a random flow of fog location distribution. This random approach is based on the work done by [33]. 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. The cloud can pay a lot with this approach without actually benefiting from the rented resources. Because of the possibility of pushing fogs anywhere without prior knowledge of the location needs, the agent might push a fog to a location with low needs, which contradicts our model objectives.

5.3 Dataset

These data were collected from a NASA server log located in Florida [14]. The data contained 13 days worth of logs from the dates between July 1 and July 13, 1995. Each row in the data indicates one user request. A sample of the data is shown in Fig. 2. Each column in these data is described as follows:

- Host: represented as an IP address
- Timestamp: Exact time when the request was received by the server. The timestamp format is DD/MON/YYYY:HH:MM:SS using a 24-hour clock. The time zone is -0400.
- Request made by the user given in quotes
- HTTP Reply code
- Bytes in the reply

For our model, the host IP address and the timestamp are only needed to make the fog placement decisions. As mentioned earlier, the first octet of the IP is used to distinguish between locations. Based on these data, there are 70 different locations. In our experiments, the preprocessing described in Sect. 4.2.2 is performed to extract the learningData matrix. For the given data, learningData[0][(12, '148')] contains:

$$[43270, 43300, 43339, 43369, 44692, 44722, 44756, 44785, 44813, \\ 44843, 44898, 44931, 44961]$$

This list contains the time in seconds of each request received by the cloud in day 0 at hour 12 and from location '146'.

5.4 Experiments setup

In the dataset, there are 70 different locations, and we chose the number of fogs that the cloud can push to be 20. This number can better show how the model can still minimize the load on the cloud by pushing the right fogs with such limited availability. The experiment setup of each objective is shown below.

- *Objective 1* For the purpose of showing the model effectiveness and ability to learn over time, we conduct the first experiment as follows. The model only reads the first-day data learns on. We then selected three days randomly from the dataset to test the model on. These days are days 2, 10, and 11. The placement decisions are made from the output of the Q-table learned on the first day only. After that, we let the agent training for all the remaining days of the dataset except days 2, 10 and 11. The purpose is to check the new decisions made by the agent following the new version of the Q-table after more learning. The results are shown and discussed in Sect. 5.5.
- *Objective 2* The next aim is to prove how our R-learning model outperforms other existing ones. This is done by implementing the threshold- and random-based approaches and comparing their decision with our model after training on several days. The day chosen for testing is day 10. The same Q-table generated from the first experiment after training on several days is used in this experiment. We set two different threshold values for the number of requests and conducted two different scenarios for each with different random-based pushing also. The threshold values are 30 and 80 requests per hour.

5.5 Evaluation objective 1—learning improvement over time

In the top three graphs of Fig. 3, each graph is represented by the hour of the day on the x-axis with respect to the number of requests received by the cloud. The green dashed line represents the number of requests sent to the cloud initially by all locations on this day. The dataset clearly shows that the peak hours or the high number of requests generated by different locations are between hours 9 and 18. Therefore, the importance of our model decision lies in this range. The red line shows the number of requests received by the cloud after the agent pushes 20 patches of fogs to locations based on our model's decision.

The top three graphs of Fig. 3 show the results of the decision made by our agent after only one day of training. Because our model starts learning randomly and because the first day was not enough to let the model learn a better pattern, you can

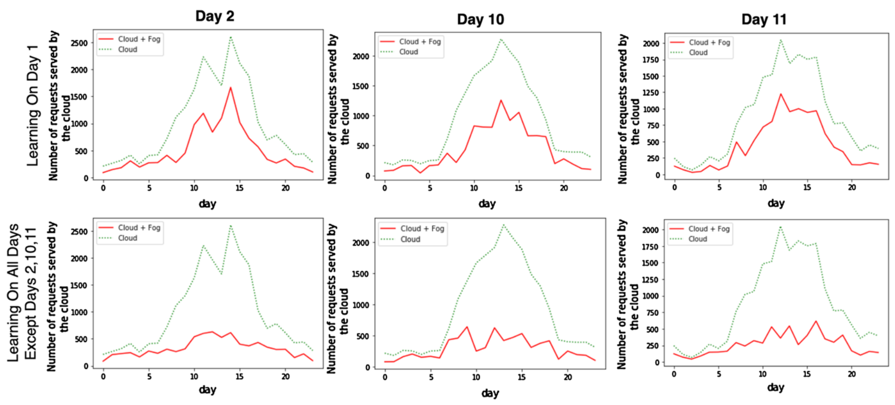


Fig. 3 Number of requests received by the cloud before and after creating fogs based on our Fog Scheduling Decision model showing the learning improvement over several days taking different testing samples

see that the improvement achieved is minimal compared to the three below graphs. We let the agent learn for the rest of the day except for days 2, 10, and 11. After learning, the agent showed more mature decisions of placing the 20 patches of fogs in better locations at a better time. This is reflected in the results of the second row of graphs shown in Fig. 3, where a load of requests sent to the cloud improved for more than 60% compared to learning on one day. It is also important to mention that less number of requests received by the cloud means the fogs are serving more requests, and therefore, the agent is doing well. Comparing the cloud load before and after the agent placement decision, we can observe that the cloud load diminished by more than 30%.

5.6 Evaluation objective 2—comparison to threshold- and random-based approaches

The results are shown in the graphs of Fig. 4. By comparing the threshold-based approach of values 30 and 100 requests per hour from a specific location, we can

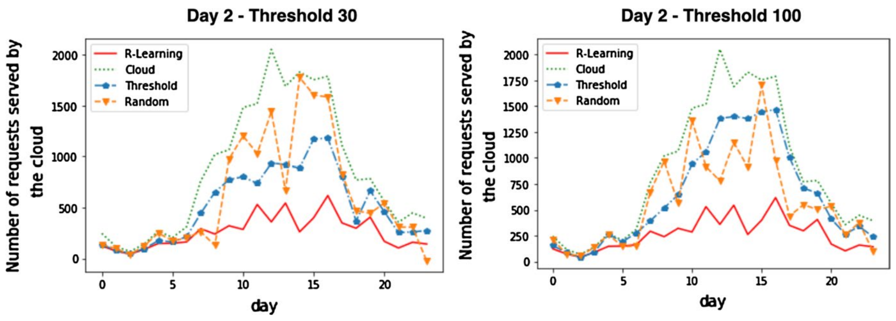


Fig. 4 Comparing R-learning decision of day 2 with the decision of threshold [27] and of random-based [33] approaches

notice that our model performs better in case of the high volume of requests, for instance, during the peak hours between hours 9 to 18. This is because of the main drawback of the threshold approach, which counts on hitting its threshold by the first location it encounters to make a decision of pushing directly. In the case of threshold 30 on Day 2, the agent searches for the first 20 locations it finds, which generates a number of requests greater than the threshold. In the case of threshold equal to 100, the performance is worse because the number of locations generating more than 100 requests per hour is most likely to be less than 20 (as per the data studied). On the other hand, in some cases, the threshold decision might do well in case the number of fogs available to push matches the number of locations generating requests more than the threshold. This is the case at hour 18 in the first graph, where we can see that the threshold model did well and similar to what our model predicts.

For the random-based approach, the agent has to select 20 random locations out of the 70 to push fogs to. Sometimes the agent can be lucky. An example can be hour 7 in the first graph. However, most of the time, the random approach is not guaranteed to produce helpful decisions. An example can be in the first graph of our experiment, where at hour 14, no fog was able to process any request coming to the cloud because they are misplaced by the random decision taken by the agent. The number of requests at this hour remained around 1750 requests, which is equal to the total number of requests received by the cloud originally. At the same hour, our model was able to minimize the number of requests to the cloud to around 400 requests. This clearly shows the advantage of our model compared to the random-based decision. In other hours of the day including the second graph of Fig. 4, we can see how the random-based model is poorly performing compared to our model.

In conclusion, even though the threshold- and random-based approaches can minimize the cloud load by approximately 70% and 90%, respectively (based on our scenarios), our model is still capable of minimizing more than 30% of this load compared to these approaches.

6 Conclusion

Creating fogs on demand to serve users and devices is a new concept that allows the creation of Kubeadm clusters containing volunteering fog nodes that provide processing and storage capabilities, which can help by spreading services provided by the cloud. This new concept is proved to increase the performance of the network by decreasing congestion, can assist in boosting the quality of service provided to the users by migrating the services to points closer to the edge, and can aid the cloud by decreasing the load pressure of incoming requests. In some cases, the cloud might have the privilege to push services on a limited number of fog devices for the purpose of minimizing the complexity of monitoring a large number of fogs and cost for volunteers that do not offer their resources for free. This motivated us to create a new reinforcement learning model that is responsible for scheduling the available clusters in locations in a way that better utilizes fogs resources. The suggested R-learning model can better predict, after giving it some time to learn the behavior of the requests, the distributions of clusters over locations at the proper time.

This model is located in the cloud, where it periodically checks the behavior of the incoming requests and updates its information using R-learning techniques. Several experiments were conducted comparing the performance of our suggested model with respect to other standard known methodologies, such as the threshold-based and random-based models for cluster distribution. As shown in the experiments, the proposed model was able to illustrate the ability to improve over time, outperform the other two methods, and achieve promising results for the time and localization problem. In this context, the number of processed requests performed by the cloud decreases from 100% to more than 30% using our model while this number decreases to only around 70% and 90% when using random-based and threshold-based approaches.

As a future direction, we are working on an extension for this work that improves the decision model to specify what services to include in the placement decision and what to keep on the cloud.

References

1. Patel KK, Patel SM et al (2016) Internet of things-IOT: definition, characteristics, architecture, enabling technologies, application and future challenges. Int J Eng Sci Comput 6(5):6122–6131
2. Gubbi J, Buyya R, Marusic S, Palaniswami M (2013) Internet of things (IOT): a vision, architectural elements, and future directions. Future Gener Comput Syst 29:1645–1660
3. Alberti AM, Singh D (2013) Internet of things: perspectives, challenges and opportunities. In: International Workshop on Telecommunications (IWT 2013), pp 1–6
4. Tordera E M, Masip-Bruin X, Garcia-Alminana J, Jukan A, Ren G-J, Zhu J, Farré J (2016) What is a fog node a tutorial on current concepts towards a common definition. arXiv preprint arXiv:1611.09193
5. Bonomi F, Milito R, Zhu J, Addepalli S (2012) 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, pp 1–6
6. Sami H, Mourad A (2018) Towards dynamic on-demand fog computing formation based on containerization technology. In: Proceedings of the 2018 International Conference on Computational Science and Computational Intelligence–CSCI'18. IEEE Computer Society
7. Kratzke N (2017) About microservices, containers and their underestimated impact on network performance. arXiv preprint arXiv:1710.04049
8. Sayfan G (2017) Mastering kubernetes. Packt Publishing Ltd, Birmingham
9. Fatema K, Emeakaroha VC, Healy PD, Morrison JP, Lynn T (2014) A survey of cloud monitoring tools: taxonomy, capabilities and objectives. J Parallel Distrib Comput 74(10):2918–2933
10. Roman R, Lopez J, Mambo M (2018) Mobile edge computing, fog et al.: a survey and analysis of security threats and challenges. Future Gener Comput Syst 78:680–698
11. Yi S, Li C, Li Q (2015) A survey of fog computing: concepts, applications and issues. In: Proceedings of the 2015 Workshop on Mobile Big Data. ACM, pp 37–42
12. Gosavi A et al (2015) Simulation-based optimization. Springer, Berlin
13. Watkins CJ, Dayan P (1992) Q-learning. Mach Learn 8(3–4):279–292
14. Nasa dataset-two months of http logs from a busy www server. [Online]. https://ita.ee.lbl.gov/html/traces.html
15. Hao Z, Novak E, Yi S, Li Q (2017) Challenges and software architecture for fog computing. IEEE Internet Comput 21(2):44–53
16. Mahmud R, Kotagiri R, Buyya R (2018) Fog computing: a taxonomy, survey and future directions. In: Internet of everything. Springer, pp 103–130
17. Tout H, Talhi C, Kara N, Mourad A (2017) Smart mobile computation offloading: centralized selective and multi-objective approach. Expert Syst Appl 80:1–13

18. Marrouche W, Harmanani HM (2018) Heuristic approaches for the open-shop scheduling problem. In: Information Technology-New Generations. Springer, pp 691–699
19. Dbouk T, Mourad A, Otrok H, Tout H, Talhi C (2019) A novel ad-hoc mobile edge cloud offering security services through intelligent resource-aware offloading. IEEE Trans Netw Serv Manag. https://doi.org/10.1109/TNSM.2019.2939221
20. Chen M, Hao Y (2018) Task offloading for mobile edge computing in software defined ultra-dense network. IEEE J Sel Areas Commun 36(3):587–597
21. Zhang J, Hu X, Ning Z, Ngai EC-H, Zhou L, Wei J, Cheng J, Hu B (2017) Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks. IEEE Internet Things J 5(4):2633–2645
22. Pham X-Q, Man ND, Tri NDT, Thai NQ, Huh E-N (2017) A cost-and performance-effective approach for task scheduling based on collaboration between cloud and fog computing. Int J Distrib Sens Netw 13(11):1550147717742073
23. Pham X-Q, Huh E-N (2016) Towards task scheduling in a cloud-fog computing system. In: 18th Asia-Pacific Network Operations and Management Symposium (APNOMS). IEEE, pp 1–4
24. Zeng D, Gu L, Guo S, Cheng Z, Yu S (2016) Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system. IEEE Trans Comput 65(12):3702–3712
25. Kherraf N, Alameddine HA, Sharafeddine S, Assi C, Ghrayeb A (2019) Optimized provisioning of edge computing resources with heterogeneous workload in iot networks. IEEE Trans Netw Serv Manag 16(5):459–474
26. Alameddine HA, Sharafeddine S, Sebbah S, Ayoubi S, Assi C (2019) Dynamic task offloading and scheduling for low-latency iot services in multi-access edge computing. IEEE J Sel Areas Commun 37(3):668–682
27. Yousefpour A, Ishigaki G, Jue J P (2017) Fog computing: Towards minimizing delay in the internet of things. In: 2017 IEEE International Conference on Edge Computing (EDGE). IEEE, pp 17–24
28. Hoque S, de Brito M S, Willner A, Keil O, Magedanz T (2017) Towards container orchestration in fog computing infrastructures. In: Proceedings of the (2017) IEEE 41st Annual on Computer Software and Applications Conference (COMPSAC), vol 2. IEEE, pp 294–299
29. Felter W, Ferreira A, Rajamony R, Rubio J (2015) An updated performance comparison of virtual machines and linux containers. In: 2015 IEEE International Symposium Performance on Analysis of Systems and Software (ISPASS). IEEE
30. Hong H-J, Tsai P-H, Hsu C-H (2016) “Dynamic module deployment in a fog computing platform. In: 18th Asia-Pacific on Network Operations and Management Symposium (APNOMS). IEEE, pp 1–6
31. Knox WB, Setapen AB, Stone P (2011) Reinforcement learning with human feedback in mountain car. In: 2011 AAAI Spring Symposium Series
32. Gosavi A (2011) A tutorial for reinforcement learning. Department of Engineering Management and Systems Engineering
33. Skarlat O, Nardelli M, Schulte S, Borkowski M, Leitner P (2017) Optimized iot service placement in the fog. Serv Oriented Comput Appl 11(4):427–443

Affiliations

Peter Farhat[1] · Hani Sami[1] · Azzam Mourad[1]

[1] Department of Computer Science & Mathematics, Lebanese American University, Beirut, Lebanon