Chapter 1

Reinforcement Learning for Service Function Chain Allocation in Fog Computing

José Santos,[1]* Tim Wauters,[1] Bruno Volckaert,[1] and Filip De Turck[1]

[1] *Department of Information Technology, Ghent University - imec, IDLab, 9052, Oost-vlaanderen, Gent, Technologiepark-Zwijnaarde 126, Belgium*

*Corresponding Author: José Santos; josepedro.pereiradossantos@UGent.be

Recently, distributed cloud infrastructures have become a potential business opportunity for most service providers due to the exponential growth of connected devices. The advent of the Internet of Things (IoT) and softwarized networks made centralized cloud systems impractical. In response, Fog Computing (FC) emerged, enabling the deployment of services on computational resources from the cloud up to the edge. However, the adoption of FC concepts is still in its early stages and challenges persist to fully benefit from fog-cloud infrastructures. One of them is known as Service Function Chaining (SFC) where providers benefit from network softwarization to create virtual chains of connected services. Recent research has tackled SFC allocation through theoretical modeling and heuristic algorithms, which often cannot cope with the dynamic behavior of the network. Thus, in this chapter, we explore a subset of Machine Learning (ML) called Reinforcement Learning (RL) to provide an efficient solution for SFC allocation in FC. The proposed approach learns about the best resource allocation decisions, focused on energy efficiency from a previously presented Mixed-integer linear programming (MILP) formulation. Results showed that RL algorithms perform comparably to state-of-the-art ILP-based implementations while offering more scalable solutions. Future research directions and open challenges are discussed.

Keywords: Fog Computing, Service Function Chaining, Reinforcement Learning, Internet of Things . . .

1.1. Introduction

With the advent of the Internet of Things (IoT), distributed cloud architectures have become a potential business opportunity for most cloud providers [1]. Low-Latency and high mobility constraints are among the strictest requirements imposed by IoT services, making centralized cloud solutions impractical. In response, cloud computing evolved towards a novel paradigm called Fog Computing (FC) [2], where a distributed cloud infrastructure is set up to provide services close to end-users. Furthermore, micro-services are currently revolutionizing the way developers build their software applications [3]. An application is decomposed in a set of self-contained containers deployed across a large number of servers instead of the traditional single monolithic application. In fact, containers are the most promising alternative to the conventional Virtual Machines (VMs), due to their low overhead and high portability. Nevertheless, several challenges in terms of resource provisioning and service scheduling persist which prevent service providers and end-users from fully benefiting from micro-service patterns. One key challenge that remains is called Service Function Chaining (SFC) [4], where services are connected in a specific order, forming a service chain that each request needs to traverse to access a particular Network Service (NS). For instance, a service chain can be composed of an API, a database and an Machine Learning (ML) service. Sensors access the API to send their data to the infrastructure while users access the database service to retrieve the sensor's collected data. This data may have already been filtered and modified by an ML service. In Fog-cloud environments, the interactions between fog locations and cloud are crucial to ensure that services operate properly due to the hierarchical architecture. For example, the database service must be allocated close to the users in a fog location, but the ML service could be instantiated in the cloud where more computing resources are available. We need proper provisioning strategies to ensure both services are allocated close enough so that users do not experience latency in accessing the inferred results. These chain requirements (e.g. service location, low-latency, minimum available bandwidth) must be guaranteed during SFC allocation in FC, which are currently not being studied since SFC concepts are still mostly unexplored in FC environments.

Although the theoretical foundations of FC have already been established, the adoption of its concepts is still in its early stages and practical implementations are still scarce. Furthermore, current studies on resource allocation are mainly focused on theoretical modeling and heuristic-based solutions, which in most cases cannot cope with the dynamic behavior of the network and leads to poor resource utilization and scalability issues. In fact, resource allocation is a difficult online decision-making problem where appropriate actions depend on fully understanding the network environment. Thus, in this chapter, we explore a subset of ML called Reinforcement Learning (RL) [5] to provide a suitable solution for SFC allocation in FC. The SFC allocation problem has been translated into an RL problem where the best resource allocation decisions (i.e. actions) are learned depending on the current status of the network infrastructure (i.e. environment). Based on a previously presented Mixed-integer linear programming (MILP) formulation, an environment has been developed where agents learn to allocate service chains in FC directly from interacting with the environment without any knowledge or information at the beginning. Our results show that RL techniques perform comparably to state-of-the-art ILP-based implementations but provide more scalable solutions.

In summary, FC is one of the most challenging topics in modern cloud computing, along with resource allocation and service chaining concepts. The rest of the chapter is organized as follows: Section 1.2 provides a brief overview of the technical background. Section 1.3 discusses the current state-of-the-art on resource allocation for FC. Section 1.4 presents the proposed RL approach for SFC allocation in FC, which is followed by the evaluation setup in Section 1.5. Next, in Section 1.6, results are shown. Finally, future research directions and open challenges are discussed in Section 1.7, concluding the chapter.

1.2. Technology overview

This section provides a brief overview of the FC paradigm. Then, the fundamental concepts related to resource allocation and service function chaining are discussed. Finally, the main concepts of RL are introduced.

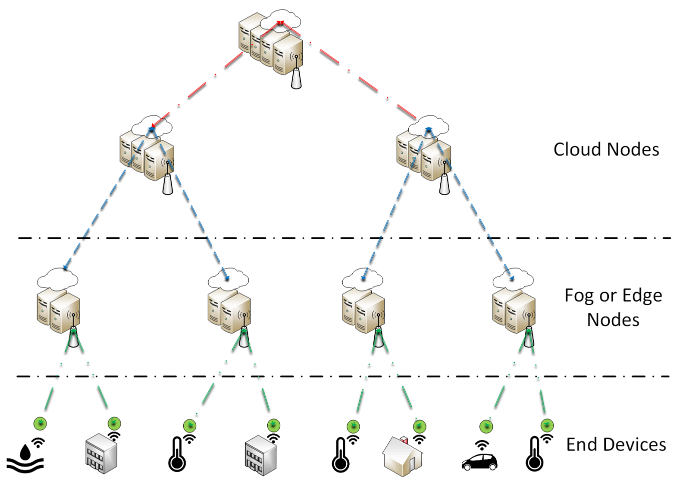


Figure 1.1: High-level view of a Fog Computing environment [6].

1.2.1. Fog Computing (FC)

The FC paradigm is an extension of cloud computing to provide resources on the edges of the network to deal with the exponential growth of connected devices [7]. Figure 1.1 presents a high-level view of the FC environment. In contrast to a centralized cloud, fog nodes are distributed across the network to act as an intermediate layer between end devices and the cloud. These so-called fog nodes, edge locations or even Cloudlets [8] are essentially small cloud entities, which bring processing power, storage procedures, and memory capacity closer to devices and end-users to enable local operations. Cloud nodes are the traditional cloud servers where a high amount of resources is available.

1.2.2. Resource Provisioning

Resource provisioning or also known as resource allocation has been studied for years in the network management domain [9, 10, 11]. Resource provisioning is related to the allocation of computing, network and storage resources needed to instantiate services requested by users and devices over the Internet. In recent years, cloud providers and users have been working together towards an efficient manner of dealing with computing resources. On one hand, users want to receive the best Quality of Service (QoS) for the minimum cost while cloud providers want to increase their revenue. Users want to maximize their service plan without increasing their costs while cloud providers want to respect the agreed QoS level by using a minimum percentage of their infrastructure. Thus, energy efficiency is essential for cloud providers while low-latency is crucial for users. Reducing costs by using the minimum amount of hardware while guaranteeing users QoS level, or increase the number of active nodes to reduce latency between the deployed service and the user to a minimum. Efficient allocation strategies are crucial for both cloud providers and users. Different provisioning policies can be applied depending on the status of the network infrastructure or the current user demand.

In addition, with the advent of FC, resource allocation has become an even more important research topic. FC has been introduced as an answer to the inherent provisioning challenges introduced by IoT services. For example, IoT services are highly challenging in terms of latency. Delay-sensitive services (e.g. connected vehicles, interactive video applications) require latencies in the order of milliseconds. If the latency increases, surpassing the communication threshold, the user connection can become unstable and the user control over the service is potentially lost. Also, since vehicles and users are continuously moving in the network area, mobility is another important factor to consider. Allocation strategies must consider service reallocations in case user connectivity is lost to ensure proper service operation at all times. Centralized cloud

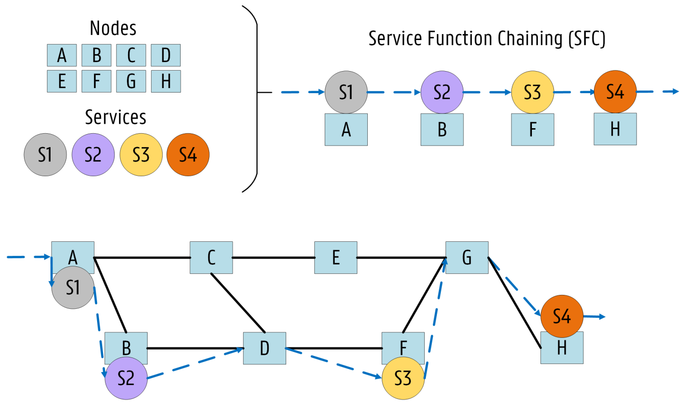


Figure 1.2: An example of a Service Function Chaining deployment [15].

infrastructures cannot fully satisfy the dynamic demands of these types of services. Therefore, FC is essential to rapidly modify the allocation of services according to highly variable demand patterns.

1.2.3. Service Function Chaining (SFC)

SFC placement [12, 13] has been studied in the network management domain during the last few years. SFC is related to the services' proper ordering ensuring that each user has to traverse the given service chain to access a particular NS as shown in Figure 1.2. The circles represent different service functions while the arrows show how the traffic is steered in the network. User requests are routed through the service chain according to a service graph, which aims to optimize resource allocation to further improve application performance. SFC enables cloud providers to dynamically reconfigure softwarized NSs without having to implement changes at the hardware level. SFC provides a flexible and reliable alternative to today's static network environment.

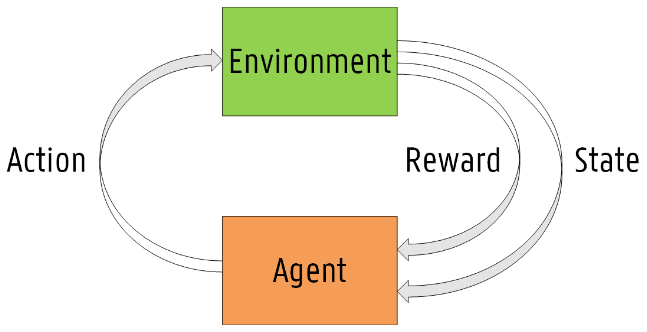


Figure 1.3: The representation schema of most RL scenarios.

1.2.4. Micro-service Architecture

Recently, micro-service patterns [14] gained tremendous attention. An application is decomposed in a set of loosely coupled services that can be developed, deployed and maintained independently. Each service is responsible for a single task and communicates with the other services through lightweight protocols. These services can then be developed in different programming languages and even using different technologies. Nowadays, containers are the most promising alternative to the traditional monolithic application paradigm, where almost everything is centralized and code-heavy.

1.2.5. Reinforcement Learning (RL)

In recent years, RL methods have become an important area in ML research [16, 17, 18]. The typical scenario in RL is represented in Figure 1.3. In most cases, RL techniques are used to solve sequential decision-making problems. An RL agent learns to make better decisions directly from experience interacting with an environment. The environment represents the problem to solve. In the beginning, the agent knows nothing about the problem at hand and learns by performing actions in an environment. For each action taken, the agent receives a reward and a new observation that describes the new state of the environment. Depending on the goal and how well the agent is performing on the given task, the reward can be positive or negative. The agent learns to be successful by repeated interaction with the environment, by determining the inherent synergies between states, actions and subsequent rewards. Ultimately, RL algorithms try to maximize the total reward an agent would collect by experiencing multiple problem rounds. For instance, let us consider an agent allocating resources in a cloud infrastructure. The agent would receive a reward for each action applied in the system. If the action translates into an appropriate allocation scheme, the agent will receive a positive reward. Otherwise, if the agent performs a bad action (e.g. terminate a service needed in the network), which produces an inappropriate allocation scheme, the reward would be negative. To maximize the reward, the agent would need to apply actions that translate into proper allocation schemes at all times. The ultimate goal in this scenario would be to train an agent able to learn good allocation actions to maximize performance and minimize costs.

1.3. State of the art

With the advent of FC concepts, efficient resource provisioning is needed in modern cloud infrastructures. This section provides a summary of the relevant previous studies concerning specifications and implementations of resource allocation strategies for fog-cloud infrastructures. First, research on resource allocation for FC is introduced, which is followed by recent advances in resource provisioning provided by ML methods. Finally, recent works on RL for resource allocation are highlighted.

1.3.1. Resource Allocation for Fog Computing

Recently, a handful of research efforts has been performed in the context of resource provisioning in FC environments. In [19], the authors proposed an allocation scheme to support crowdsensing applications in the context of IoT. Their approach has been formulated as a MILP model which takes cost-efficient provisioning and task distribution into account. Results confirmed that their proposal could outperform traditional cloud infrastructures. In [20], an optimization formulation for the QoS-aware deployment of IoT applications over Fog infrastructures has been proposed and implemented as a prototype called FogTorch. Their approach focused not only on hardware and software demands but also on QoS requirements, such as network latency and bandwidth. Results showed that their algorithm ensures optimal service deployment while decreasing hardware capacity and increasing resource demands. Additionally, in [21], the IoT resource allocation problem in FC has been modeled as an ILP formulation, where QoS metrics and deadlines for the deployment of each application have been considered. Results proved that their formulation can prevent QoS violations and reduce costs when compared to a traditional cloud approach. Furthermore, in [22], a particle swarm optimization algorithm has been proposed for the resource allocation problem in fog-cloud infrastructures specifically focused on Smart buildings. Results showed that their approach can reduce the response time and the cost of VM allocations. In [23], an ILP model for the Fog resource provisioning problem has been formulated followed by a heuristic-based algorithm able to find suboptimal solutions, albeit achieving better time efficiency. In their work, the authors studied the trade-off between maximizing the reliability and minimizing the overall system cost. Moreover, in [24], service placement strategies for FC based on matching game algorithms have been introduced. On one hand, the first approach is based on SFC concepts since the ordered sequence of services requested by each application is considered. On the other hand, the second one formulates the problem while overlooking the chain structure to lower the computation complexity without compromising performance. Also, in [25], an edge container orchestrator for low powered devices called FLEDGE has been presented. Their results showed that FLEDGE minimizes resource costs when compared with other platforms. Recently, their work has been extended in [26], where the scalability and volatility of a fog-cloud infrastructure have been studied. The authors proposed a scheduling algorithm to allocate services in a large-scale Fog deployment capable of adapting to network changes.

Although most of the cited research has dealt with allocation issues in FC, none of the aforementioned studies considered realistic QoS requirements or any kind of constraints coming from the high demands imposed by IoT (e.g. latency thresholds, service location, container-based services). Furthermore, most research is focused on theoretical modeling and simulation studies which limit their practical implementation.

1.3.2. ML techniques for Resource Allocation

Due to recent advances in ML, studies have been carried out to apply ML methods to resource allocation problems. In [27], the authors proposed supervised learning techniques to predict future NFV requests. Their goal is to proactively allocate resources based on previously observed patterns. Results showed that their proposal can proactively satisfy NFV requests. In [28], neural-network models have been proposed to address the VNF auto-scaling problem in 5G Networks. Their goal is to predict the required number of VNFs at a given moment based on previous traffic demands. Furthermore, an ILP formulation has been presented to solve the SFC allocation problem. Results proved that the average end-to-end (E2E) latency reduces significantly when service chains are allocated at the edge. In [29], an ML model has been employed to predict VNF resource demands with high accuracy (e.g. CPU). Their approach can be applied to SFC allocation problems, such as auto-scaling and optimal placement. Additionally, in [30], the fog infrastructure has been modeled as a distributed intelligent processing system called SmartFog by using ML techniques and graph theory. Their approach provides low-latency decision-making and adaptive resource management through a nature-inspired fog architecture.

In summary, supervised and unsupervised ML techniques have been implemented in the literature to improve decision-making in cloud infrastructures. Most cited research deals with allocation and auto-scaling problems that traditional methods (e.g. theoretical modeling, heuristic algorithms) have not been able to fully resolve due to the dynamic behavior of the network.

1.3.3. RL methods for Resource Allocation

Recently, RL methods have been given significant attention in the field of resource allocation. In [31], a deep RL technique called DeepRM has been presented to solve the task placement problem in a cloud management system. Their initial results showed that DeepRM performs comparably to heuristicsbased solutions and that it can learn different strategies depending on the network status. In [32], the IoT service allocation problem is addressed by employing an RL mechanism to calculate satisfactory levels of Quality of Experience (QoE). Their evaluations proved the efficiency of the applied methods. Furthermore, in [33], an RL-based optimization framework has been presented to tackle the resource allocation problem in wireless Multi-access edge computing (MEC). Their objective is to minimize costs while optimizing resources. Simulation results have been presented where their proposed methods achieved significant cost reductions. In [34], RL techniques have been studied for their applicability to the SFC allocation problem in NFV-SDN enabled metro-core optical networks. Their results demonstrated the advantages of using RL-based optimizations over rule-based methods. Additionally, in [35], a fog resource scheduling mechanism based on deep RL has been presented. Their approach is focused on vehicular FC use cases aiming to minimize the time consumption of safety-related applications. Results showed that their proposed schemes can reduce time consumption when compared to traditional approaches.

In summary, RL methods have proven their potential applicability to resource allocation issues during the past years. However, the performance of RL techniques is deeply interconnected with the way the environment and the reward system is set up. Depending on the assumptions made in the system, RL methods could deliver completely different results. To the best of our knowledge, RL methods have not yet been applied to SFC allocation where fog-cloud infrastructures and container-based services have been assumed. Also, the dynamic behavior of the network and different scheduling strategies (e.g. low-latency, energy efficiency) have not been entirely addressed. However, RL techniques have proven that learning directly from experience could work in a practical deployment and offer a real alternative to current heuristic-based approaches. Thus, in the next section, a novel RL approach for SFC allocation in FC is proposed.

1.4. A RL approach for SFC Allocation in Fog Computing

This section introduces the RL approach for SFC allocation in FC. First, the IoT allocation problem formulation is presented. Then, the observation and action spaces from our RL environment are described. Finally, the reward function and the agent are introduced.

1.4.1. Problem formulation

As mentioned, the IoT allocation problem has been modeled as a MILP formulation previously presented in [36]. The model considers a fog-cloud infrastructure where containerized service chains can be allocated. An IoT application is decomposed in a set of micro-services, which have a particular replication factor for load balancing or redundancy. Multiple users are expected to access these micro-services. The fog-cloud infrastructure manages a set of nodes, in which micro-service instances must be allocated based on its requirements and subject to multiple constraints. For example, nodes have limited capacities (e.g. CPU and memory) and all micro-services composing a given application must be allocated in the network so that the application can be considered deployed. The MILP formulation has been translated into an RL environment

Table 1.1: Variables used for the minimization of the overall system cost.

Symbol	Description
$P^{a,id}_{s,\beta_i}(n)$	The placement matrix. If $P^{a,id}_{s,\beta_i}(n) = 1$, the replica β_i of micro-service s is executed on node n for the application a with the SFC identifier id.
ϖ_n	The associated weight to node n
ω_s	The CPU requirement (in cpu) of the micro-service s
γ_s	The memory requirement (in GB) of the micro-service s
δ_s	The bandwidth requirement (in Mbit/s) of the micro-service s.

called gym-fog [1] where actions can be performed and at each time step, a new observation is given which describes the new state of the environment. It should be noted that only the cloud formulations have been considered for the designing of the RL approach and wireless aspects available in the model have not been used.

For this work, we designed a new objective for the MILP model: the minimization of the overall cost of the system, which translates into increased energy efficiency. By using the nomenclature of the MILP formulation presented in Table 1.1, this objective can be expressed as shown in (Equation 1.1). The agent will try to learn how to minimize the overall system cost as a MILP formulation by interacting with the gym-fog environment.

$$\sum_{a \,\varepsilon\, A} \sum_{id \,\varepsilon\, ID} \sum_{s \,\varepsilon\, S} \sum_{\beta_i \,\varepsilon\, \beta} \sum_{n \,\varepsilon\, N} P^{a,id}_{s,\beta_i}(n) \times \varpi_n \times \omega_s \times \gamma_s \times \delta_s \qquad (1.1)$$

[1]https://github.com/jpedro1992/gym-fog

Table 1.2: A sample fraction of the observation space of the gym-fog environment.

Metric Name	Description
Ratio $S1$ (RS1)	The relation of allocated micro-service 1 instances between the MILP model and the agent.
Ratio $S2$ (RS2)	The relation of allocated micro-service 2 instances between the MILP model and the agent.
Cost RL (RL)	The agent allocation scheme cost.
Cost MILP (MILP)	The MILP allocation scheme cost.
User Requests (UR)	The number of user requests at the given moment.

1.4.2. Observation Space

The observation space corresponds to the state representing the environment at a given step. For instance, consider an agent playing a chess game, the observation will be the board status of a particular game. In the implemented gym-fog, the observation space has been designed as shown in Table 1.2. For an easier understanding of our methodology, let us consider a small infrastructure where all user requests coming to our system are made based on an IoT application decomposed in 2 individual micro-services. The observation space will be constituted by five metrics. Two metrics (RS1 and RS2) represent the ratio between the allocation scheme proposed by the agent and the MILP model for each micro-service. Then, two metrics (RL and MILP) represent the overall system cost given by the agent and the MILP model, respectively. Finally, the last metric (UR) is about the exact number of user requests made in the network at that particular step. Thus, the observation space increases linearly with the number of applications available in the MILP model and their corresponding micro-services.

Table 1.3: A sample fraction of the action space of the gym-fog environment.

Action Label	Description
DoNothing	The agent does nothing.
Deploy-S1-N1	Allocate a micro-service 1 instance in node 1.
Deploy-S2-N1	Allocate a micro-service 2 instance in node 1.
Stop-S1-N1	Terminate the micro-service 1 instance in node 1.
Stop-S2-N1	Terminate the micro-service 2 instance in node 1.

1.4.3. Action Space

The action space corresponds to all actions that the agent could apply in the environment. Considering the same chess analogy as before, the action space in a chess game would be selecting each piece and move it to a certain board position. In a fog-cloud infrastructure, the action space must include the allocation and termination of all micro-services available in the system. The action space of the gym-fog environment has been designed as shown in Table 1.3 assuming the same IoT application constituted by 2 micro-services and that the fog-cloud infrastructure is represented by only one node. The action space is composed of 5 discrete actions. The action space also increases linearly with the number of micro-services and the number of nodes available in the infrastructure. The first action is called as *DoNothing* since when applied by the agent, no allocation or termination will be performed in the network. Thus, the agent should only select this action when the current allocation scheme meets the current network demand. The second set of actions corresponds to the allocation of micro-service instances (Deploy-Si-Ni). The Agent can choose which micro-service instance should be allocated and on which node it should be executed. The action space has been designed in this manner to make sure that the agent can find better allocation decisions by choosing a particular micro-service instance to be deployed on a certain node. The agent can apply a given action from this set several times if more instances of the same micro-service are needed in the network to support all user requests. Finally, the last set of actions (Stop-Si-Ni) corresponds to the termination of micro-service instances. As in allocation actions, the agent chooses which micro-service should be terminated and which instance should it be since the node where the micro-service is deployed is also given. Our goal is to teach the agent that a certain number of micro-service instances needs to be allocated for the proper chain operation and to support all user requests.

1.4.4. Reward Function

The purpose of the reward function is to teach the agent how to maximize the accumulated reward by selecting appropriate actions depending on the observation provided by the environment. A certain reward is obtained for each action the agent selects. This reward can be positive or negative. Thus, the agent can learn if its chosen action was appropriate based on the received reward. The design of an appropriate reward function through the manual tuning of ML parameters is needed to ensure the agent learns what it is supposed to. The reward function implemented in the gym-fog environment is shown in Alg. 1. The agent's purpose is to learn how to allocate micro-services in a fog-cloud infrastructure according to the MILP formulation. The MILP model provisions services in the network area by minimizing the overall system cost, as shown previously. Therefore, the closer the agent is to achieve the MILP's solution, the higher the reward it receives. First, rewards are calculated based on constraints included in the MILP model. For instance, a constraint has

Algorithm 1 Reward Function of the gym-fog environment

Input: Observation state after action step in
Output: Reward out

```
// Return the reward for the given state
getReward(obs):
    $reward = 0$
    $ratioS1 = obs.get(1)$
    $ratioS2 = obs.get(2)$
    $costRL = obs.get(3)$
    $costMILP = obs.get(4)$

    // Reward based on Keywords for MILP constraints
    // Constraint: MAX micro-services on a single Node
    // Constraint: Terminate micro-service without deployment first
    // Constraint: MAX micro-service instances reached
    if $constraintMaxServicesOnNode == True$ then
        return $-1$
    if $constraintTerminateServiceFirst == True$ then
        return $-1$
    if $constraintMAXServiceInstances == True$ then
        return $-1$

    // Micro-service ratio Reward calculation
    $reward = reward + getRatioReward(ratioS1)$
    $reward = reward + getRatioReward(ratioS2)$

    // Cost Reward Calculation
    $reward = reward + getCostReward(costRL, costMILP)$

    // Ultimate Goal calculation
    if $ratioS1 == 1$ and $ratioS2 == 1$ then
        if $costRL > costMILP$ then // High Reward
            $reward = reward + 10$
        if $costRL == costMILP$ then // MAX Reward
            $reward = reward + 100$

    return $reward$
```

Algorithm 2 Micro-service Ratio Reward Calculation

Input: Micro-service Ratio observation state in
Output: Ratio reward out

```
// Return the reward for the given micro-service ratio
getRatioReward(ratio):
    if $ratio == 0$ then // No service deployed - Bad solution
        return $-5$
    else if $ratio == 1$ then // Equal to the MILP - Good solution
        return 5
    else then // Under / Over-provisioning scheme
        return $-1$
```

Algorithm 3 Cost Reward Calculation

Input: CostRL, CostMILP in
Output: Cost reward out

```
// Return the reward for the relation between the CostRL and CostMILP
getCostReward(costRL, costMILP):
    if $\text{costRL} < \text{costMILP}$ then // Lower than MILP - Bad solution
        return $-10$
    else if $\text{costMILP} \leq \text{costRL} \leq 1.10\times\text{costMILP}$ then // Best Solution
        return 10
    else if $1.10\times\text{costMILP} < \text{costRL} \leq 1.25\times\text{costMILP}$ then
        return $-2$
    else if $1.25\times\text{costMILP} < \text{costRL} \leq 1.75\times\text{costMILP}$ then
        return $-4$
    else if $1.75\times\text{costMILP} < \text{costRL} \leq 2.0\times\text{costMILP}$ then
        return $-6$
    else if $2.0\times\text{costMILP} < \text{costRL} \leq 3.0\times\text{costMILP}$ then
        return $-8$
    else if $3.0\times\text{costMILP} < \text{costRL} \leq 4.0\times\text{costMILP}$ then
        return $-10$
    else then // $> 4\times costMILP$
        return $-20$
```

been added to limit the allocation of one instance of the same micro-service per node. Thus, if the agent selects an action that would revoke this constraint, the agent would receive a negative reward (i.e. -1). Then, individual rewards are calculated for each micro-service ratio as shown in Alg. 2. First, if the number of allocated micro-service instances by the agent is equal to zero, a reward of -5 is retrieved because the agent is not allocating a single instance of this micro-service, which prevents the service chain from proper operation. Second, if the number of allocated micro-service instances by the agent is equal to the ones allocated by the MILP model, a reward of 5 is returned. Finally, a reward of -1 is retrieved in case the agent is allocating a higher number of replicas that are not required (i.e. over-provisioning) or if the agent is allocating fewer instances than needed (i.e. under-provisioning).

After ratio reward calculation, a cost reward function is performed as shown in Alg. 3. First, if the agent's cost is lower than the MILP one, a negative reward is returned because the agent cannot have a lower cost since the MILP solution is optimal. Thus, the agent is probably violating several constraints of the IoT service problem. Then, if the agent's cost is equal or up to 10% higher than the MILP one, 10 is returned since the agent is performing similar to the MILP model. Then, depending on how higher the agent cost is compared to the MILP one, a decreasing reward is returned, meaning that the agent is being taught that the closer it stays to the MILP cost, the higher reward it receives. Nevertheless, the ultimate goal is to achieve similar costs to the MILP model and allocate all necessary micro-service instances for the acceptance of all user requests. Thus, two bonus rewards can be given to the agent if all micro-service ratios are equal to 1. First, if the agent's cost is higher than the MILP cost, a bonus reward of 10 is given since the agent allocated all micro-service instances needed in the network, despite the higher cost. Second, if the agent's cost matches the MILP cost, a bonus reward of 100 is retrieved because the agent accomplished exactly what it was supposed to. The agent learned how to allocate micro-services in a fog-cloud infrastructure as a MILP formulation.

1.4.5. Agent

This section introduces the Q-Learning agent used in the evaluation of the gym-fog environment. Q-learning [37] is a classical RL algorithm that learns the best action to select at a given state by experiencing each state-action pair $Q(s, a)$. Q-learning is an off-policy RL method since the agent learns the optimal policy (π) independently of the applied actions based on a two-step process. The first process is called **exploitation** where a Q-table is calculated as a baseline for all possible actions for a given state. Then, the action with a higher value (i.e. maximum reward) would be applied. The second operation is called **exploration** since the agent instead of selecting actions based on the maximum future reward, the agent selects an action at random which allows the exploration and discovery of new states that otherwise could have not been explored due to the exploitation process. Exploration and exploitation rates can be settled at run time, thus complete control over the algorithm is provided.

The main issue with Q-learning agents is that it requires to see all action-state pairs for a given environment to be able to apply actions that would maximize reward. As the problem size grows, representing all state-actions pairs in memory becomes prohibitive. For instance, increasing the complexity

Table 1.4: The reduced observation space of the gym-fog environment.

Metric Name	Number of States
RatioS1	3 states (ratio calculation): $[RS1 = 0, RS1 = 1, else]$
RatioS2	3 states (ratio calculation): $[RS2 = 0, RS2 = 1, else]$
Cost	8 states (cost calculation): [RL $<$ MILP, MILP $\leq$ RL $\leq$ 1.10$\times$MILP , ... , RL $>$ 4.0$\times$MILP]
UserRequests	4 states: $[UR <= 20, UR <= 32, UR <= 40, UR <= 50]$
Total	288 states ($3 \times 3 \times 8 \times 4$)

of the gym-fog environment (e.g. adding nodes to the infrastructure, adding extra services to the service chain), has a serious impact on memory and execution time because it is directly linked with the size of the action and state space. Thus, to reduce the space complexity, the observation space has been discretized as shown in Table 1.4, where a specific range for each observation metric has been attributed reducing significantly the number of states that the Q-learning agent needs to consider. Assuming the previous fog-cloud infrastructure, the observation space would have been reduced into 288 discrete states. First, the observation metrics regarding micro-service allocations have been reduced into 3 spaces. For instance, the *Ratio* $S1$ can only be equal to 0, equal to 1, or anything else (i.e. all other possibilities are grouped). These 3 states are the only states that the Q learning agent needs to consider to find good actions regarding the *Ratio* $S1$ metric. Additionally, the two cost observation metrics (costRL and costMILP) have been combined into a new metric called cost where the difference between these two is used to formulate 8 states based on the previously shown cost reward function. Finally, user requests are also aggregated into 4 states based on the solutions provided by the MILP model, which vary depending on the service chains to be allocated and on the considered fog-cloud infrastructure.

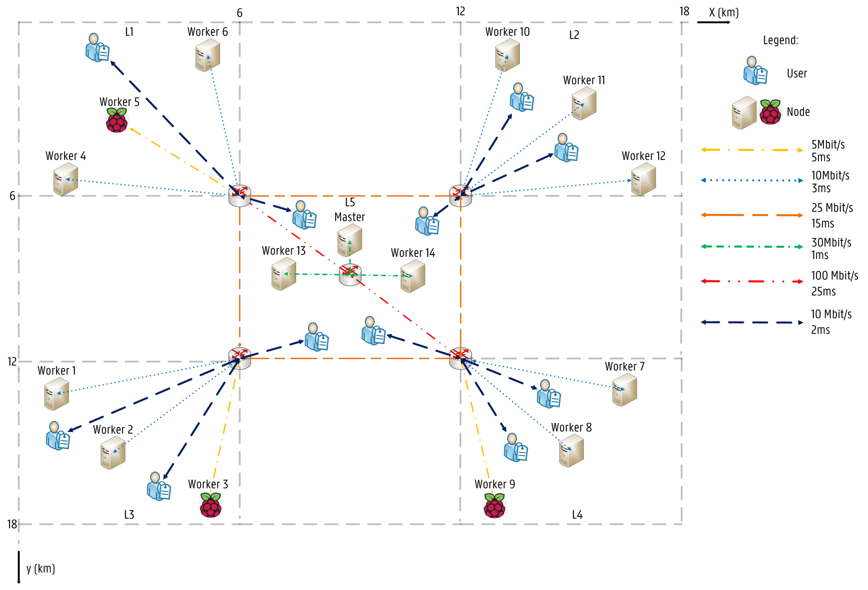


Figure 1.4: The fog-cloud infrastructure for the gym-fog environment evaluation.

1.5. Evaluation Setup

This section describes the fog-cloud infrastructure used for the evaluation of the gym-fog environment. Then, the environment implementation is detailed followed by the respective configuration applied in the evaluation.

1.5.1. Fog-cloud infrastructure

The fog-cloud infrastructure illustrated in Fig. 1.4 has been represented in the gym-fog environment. A total area of 324 km^2 has been considered. The fog-cloud infrastructure is deployed on five locations L, where the micro-service allocation is possible. Each location manages a set of three nodes. The hardware configurations of each node are shown in Table 1.5. Each node has a given computing capacity (i.e. CPU, RAM and Bandwidth) and a certain weight,

Table 1.5: The hardware configuration of each node.

Node	CPU (cpu)	RAM (Mi)	Band. (Mbit/s)	Weight
Worker 1	2.0	4.0	10.0	2.0
Worker 2	2.0	4.0	10.0	2.0
Worker 3	1.0	2.0	5.0	1.0
Worker 4	2.0	4.0	10.0	2.0
Worker 5	1.0	2.0	5.0	1.0
Worker 6	2.0	4.0	10.0	2.0
Worker 7	2.0	4.0	10.0	2.0
Worker 8	2.0	4.0	10.0	2.0
Worker 9	1.0	2.0	5.0	1.0
Worker 10	2.0	4.0	10.0	2.0
Worker 11	2.0	4.0	10.0	2.0
Worker 12	2.0	4.0	10.0	2.0
Worker 13	6.0	16.0	30.0	3.0
Worker 14	6.0	16.0	30.0	3.0
Master	8.0	24.0	30.0	3.0

which are the necessary information to calculate the overall system cost based on the MILP formulation.

1.5.2. Environment Implementation

The gym-fog environment was developed based on the OpenAi gym [38]. OpenAi gym is an open-source toolkit for RL research written in Python. It includes a collection of benchmark problems that expose a standardized interface comparing RL algorithms in terms of performance. The MILP formulation initially developed in Java has been rewritten in Python to ease the interaction between the MILP model and the OpenAi gym. The gym-fog environment was built based on the OpenAi gym structure as shown in Figure 1.5. To begin the ex-

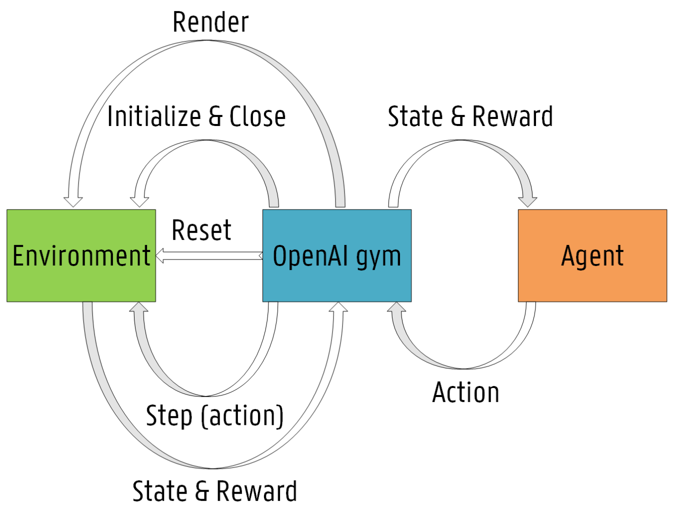


Figure 1.5: The OpenAi gym environment structure.

periment, the initialize function is triggered. Then, during the training, at each iteration, the agent selects an action which then is passed to the environment by OpenAi gym through a step function, where a new state and the respective reward are returned. Also, a reset function is used at the beginning or when each episode is finished so that the initial state of the environment is reinstated. Furthermore, a render function can be used to render the environment after each step. Finally, a close function is called when the learning process is completed to properly terminate the environment. The implemented gym-fog uses the mentioned functions to interact with the MILP model. Essentially, OpenAi gym acts as a bridge between our MILP model and the agent.

1.5.3. Environment Configuration

The gym-fog environment configuration is shown in Table 1.6 based on the described fog-cloud infrastructure. One application is available, which is de-

Table 1.6: The gym-fog environment configuration.

Name	**Description**
Number Applications	1
Number of Micro-services	3
Number of Locations	5
Number of Nodes	15
The SFC Structure	$a_1 : s_1 \rightarrow s_2 \rightarrow s_3$
Max. Replication factor	5
Action Space	91 actions
Observation Space	6 Observation Metrics
Reduced Observation Space	864 states
Episode Duration	100 steps
Agent Explore / Learning Rate	0.01 / 0.001

composed in three micro-services composing a service chain. The maximum replication factor corresponds to 5, meaning that the MILP model or the agent can only deploy 5 micro-service instances of the same type of micro-service. The action space is constituted by 91 actions (3 micro-services, 15 nodes), while the observation space is constituted by 6 observation metrics which have been reduced into 864 discrete states. Each episode duration is constituted of 100 steps. The agent's explore rate and learning rate have been settled to 0.01 and 0.001, respectively. For the evaluation, the agent and the MILP calculations have been executed on a 6-core Intel i7-9850H CPU @ 2.6 GHz processor with 16 GB of memory.

1.6. Results

This section presents the obtained results. First, a static scenario has been evaluated where the number of user requests is kept constant throughout the evaluation. Then, a dynamic use case is assessed where the network demand is constantly changing since users join and leave randomly.

1.6.1. Static scenario

As a first evaluation, the Q-learning agent has been trained during 10000 episodes by considering a static use case where the number of user requests has been kept constant. Thus, dynamic changes in terms of user requests are not expected in this experiment. The Q-learning agent should be able to learn significantly faster adequate actions in this use case than in a dynamic scenario since the number of requests is constant throughout all training. In Figure 1.6, both the reward accumulated and the cost difference between the Q-Learning agent and the MILP model are illustrated. A smoothing window of 100 has been applied to reduce spikes in both curves. As shown, the agent can reduce the overall system cost reaching solutions 5% worse than the MILP model. Additionally, accumulated rewards of 1200 have been obtained in a single episode meaning that the agent is receiving on average a reward of 12 per step, which based on our implemented reward function means that the agent is allocating all the required micro-service instances in the infrastructure though it is not able to fully optimize the overall system cost as the MILP model. Also, another important factor to consider is the percentage of accepted requests in each episode shown in Figure 1.7 since the agent can reach low costs without allocating all the necessary micro-services which would translate into unaccepted user requests. The first 10 steps have been disregarded regarding the acceptance of requests as a warming up period, ensuring that the agent has enough steps to properly select actions. As shown, the Q-learning agent can accept all

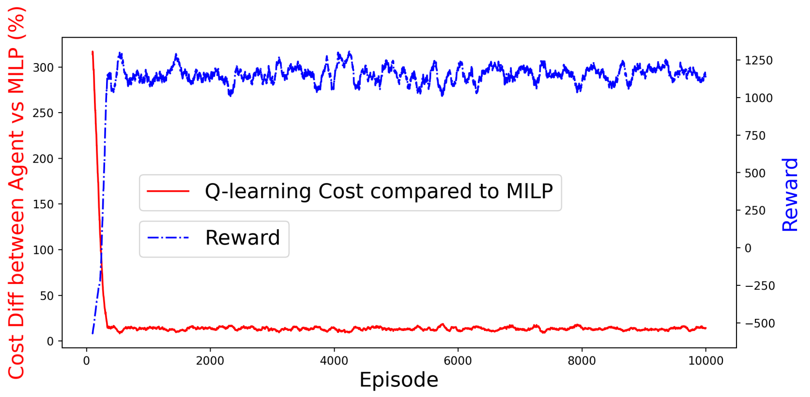


Figure 1.6: The accumulated reward and the cost difference for the static use case.

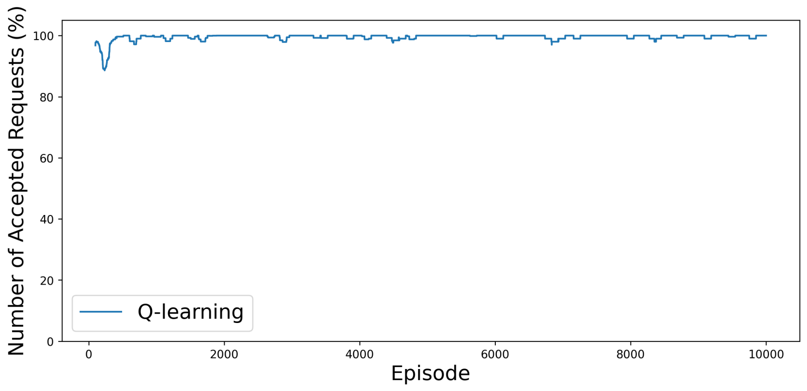


Figure 1.7: The percentage of accepted requests for the static scenario.

user requests (i.e. 100%) consistently after 500 episodes. Finally, in Figure 1.8, the execution time of each episode is presented. The Q-learning agent solves a single episode in on average 0.15 and 0.20 seconds, which compared to ILP-based calculations is significantly faster because an ILP formulation needs to calculate the optimal allocation scheme on each episode step. Results prove that the Q-learning agent is not only able to learn allocation schemes with low costs, but also accept all user requests for a static scenario.

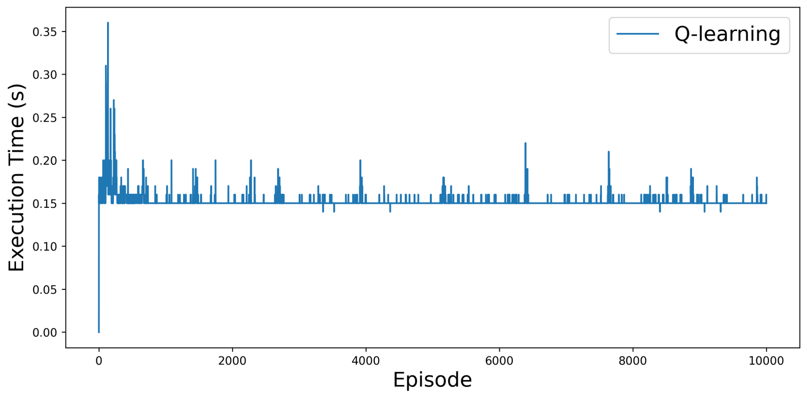


Figure 1.8: The execution time of each episode run.

Table 1.7: The MILP model execution time.

	Number User Requests							
	1	5	10	20	25	30	40	50
Execution Time (s)	0.05	0.12	0.20	0.27	5.09	5.95	9.33	48.83

1.6.2. Dynamic scenario

In the dynamic scenario, the network demand is constantly changing during the episode. The number of user requests may decrease or increase and the agent must adapt its allocation scheme according to the network demand. The number of user requests has been changed every 5 steps between 1 and 50 based on specific probabilities (increase: 50%, equal: 35%, decrease: 15%). The total increase or decrease is random, which makes this dynamic scenario more challenging than the previous static case since no pattern is given to the agent throughout the experiment since several patterns occur in different episodes. In Table 1.7, the MILP execution time for each configuration is shown. For instance, for user requests higher than 25, the MILP model requires at least 5 seconds to obtain the optimal allocation scheme. For even higher values of user requests, the MILP model takes on average at least 10 seconds. These calculations represent a single step on an episode, which proves the drawback of ILP-based solutions because every change on the network, would require a new calculation making these solutions impractical. In Figure 1.9, both the accumulated reward and the average cost difference between the Q-Learning agent and the MILP model are illustrated. The Q-Learning agent can reduce the overall system cost reaching solutions 50% worse than the MILP model. Additionally, the agent only accumulates rewards of 300 in a single episode, meaning that the agent is receiving on average a reward of 3 per step. Based on our implemented reward function, this means that the agent is not able to allocate all the required micro-service instances in the infrastructure, affecting the percentage of accepted requests as shown in Figure 1.10. Due to the dynamic demand, the agent needs to constantly adapt the allocation scheme in the infrastructure, which translates into under-provisioning and over-provisioning schemes during several steps in a single episode. Thus, the acceptance of requests oscillates between 40% and 90% during the 10000 episode training when a smoothing window of 10 is applied. The agent is constantly reacting to demand changes, which makes this scenario notably more challenging than the previous static use case. Results prove that efficient solutions for dynamic resource allocation are still missing due to the problem complexity. It is hard to find practical approaches that meet user demands while decreasing the overall system cost. Nevertheless, our early results show that RL can be applied to SFC allocation in FC and should be further explored in future research. The extension of the observation space is left for future work as it could improve these cost and acceptance results.

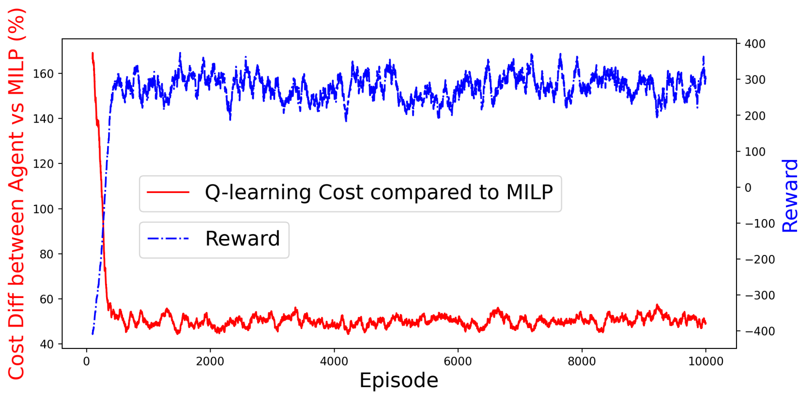


Figure 1.9: The accumulated reward and the cost difference for the dynamic case.

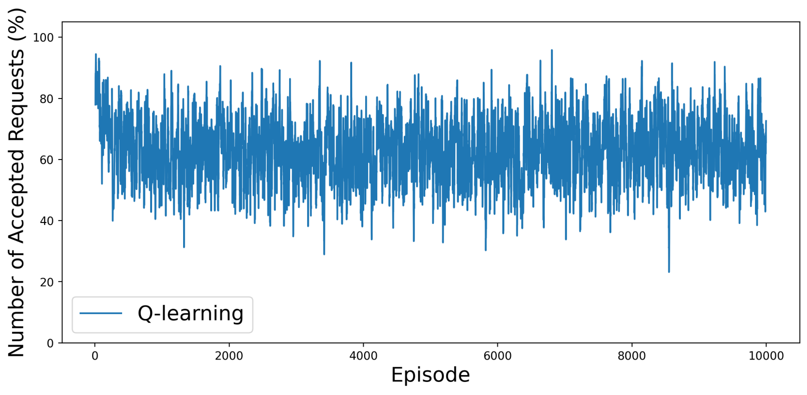


Figure 1.10: The percentage of accepted requests for the dynamic scenario.

1.7. Conclusion and future direction

Over the past years, ML techniques have become an interesting research field in the networking domain. Several efforts have been made to adapt ML methods to common network problems. This chapter focuses on RL agents to provide an efficient solution for SFC allocation in fog-cloud infrastructures. Resource provisioning has been studied for years in the network management field. However, networks and services are continuously evolving, with new protocols and technologies introduced to address current problems and improve the overall QoS. Recent examples include the adoption of SFC, micro-service paradigms and FC. Services are connected in a specific order to improve flexibility and resource allocation performance. Also, the micro-service pattern revolutionized the way developers are currently building their software applications. An application evolved from a single monolithic into a set of small containers, which may be deployed across several servers. Thus, traditional centralized clouds evolved into small distributed fog locations to distribute computing resources across the network area. And when all these concepts come into place, resource allocation is a quite complex online task. Research provisioning research in fog-cloud infrastructures is still in its early stages. Distributing the infrastructure has increased operational costs for service providers and energy consumption has become a growing concern. We addressed this challenge in this chapter by employing RL agents to find proper allocation decisions, focused on reducing the overall system cost. An environment called gym-fog has been developed to bridge the gap between ILP-based solutions with RL algorithms. Observation and action spaces have been designed for the resource allocation problem to teach RL agents how to allocate services in FC. A reward system has been set up to incentivize RL agents to select appropriate actions for SFC allocation focused on reducing the overall system cost, translating into higher energy efficiency. Results proved that our developed agent can obtain comparable performance to state-of-the-art ILP formulations for static use cases, where 100% of requests have been accepted with overall costs 5% worse than our MILP model. In contrast, dynamic use cases also proved their complexity by showing that practical solutions able to reduce the overall cost and accept all user-requests are still missing. Our agent can reduce costs up to 50% and accept on average 60% of the requests.

Developing RL systems able to learn directly from experience without any prior knowledge and capable of reallocating services in the infrastructure by reacting to sudden network changes will be the next main topic in this research field. RL methods have already proven their potential applicability to the resource provisioning domain. However, the performance of these techniques is deeply interconnected with the way the RL system is setup. The environment is the key to the problem. The interactions between the agent and the environment affect greatly the performance of these algorithms. Further, the state and action space of the problem can grow exponentially depending on the infrastructure size (i.e. the number of nodes, the number of services) used in the environment, which could lead to an unsolvable problem. Finally, the importance of the reward system should not be neglected. The agent will only learn to properly allocate services if it is compensated with positive rewards during the learning process, even if it was not able to reach desirable solutions. The key is to give the agent higher rewards the closer it is to reach the ultimate goal, otherwise, it is quite challenging for the agent to learn proper actions. In contrast, ILP-based methods are difficult to implement in practice due to their resolution time. Also, they require a lot of initial information to be fed to the algorithm so that optimal allocation schemes can be found. These methods can take hours or even days to find the optimal service allocation and when network changes occur, service reallocations should be made as fast as possible. Another challenge is the lack of expertise in both RL and resource allocation fields. Few experts have significant knowledge in both domains, which makes it difficult to implement RL solutions adapted for resource allocation problems. Most RL methods used in networking have been created for other types of applications (e.g. video games).

In summary, several challenges persist in the resource allocation domain. Nevertheless, given the dynamic behavior of the network and the need for efficient scheduling strategies (e.g. energy efficiency, low-latency), RL methods have proven that with enough training they can be an adequate solution for resource provisioning in fog-cloud infrastructures. Furthermore, these methods have shown their potential in practical scenarios where current ILP-based solutions have several drawbacks, especially in terms of scalability. As future work, we will extend our gym-fog environment by designing more complex reward functions capable of fully addressing the challenges of dynamic use cases, as well as experiment with different RL agents as deep queue networks and actor-critic methods.

Bibliography

[1] Biswas, Abdur Rahim and Giaffreda, Raffaele. IoT and cloud convergence: Opportunities and challenges. *IEEE World Forum on Internet of Things (WF-IoT)*, 375–376, 2014.

[2] Naha, Ranesh Kumar and Garg, Saurabh and Georgakopoulos, Dimitrios

and Jayaraman, Prem Prakash and Gao, Longxiang and Xiang, Yong and Ranjan, Rajiv. Fog Computing: Survey of trends, architectures, requirements, and research directions. *IEEE access*, 6:47980–48009, 2018.

[3] Newman, Sam. Building microservices: designing fine-grained systems. *" O'Reilly Media, Inc."*, 2015.

[4] Bhamare, Deval and Jain, Raj and Samaka, Mohammed and Erbad, Aiman. A survey on service function chaining. *Journal of Network and Computer Applications*, 75:138–155, 2016.

[5] Sutton, Richard S and Barto, Andrew G. Reinforcement learning: An introduction. *MIT press*, 2018.

[6] Santos, José and Wauters, Tim and Volckaert, Bruno and De Turck, Filip. Fog computing: Enabling the management and orchestration of smart city applications in 5g networks. *Multidisciplinary Digital Publishing Institute*, 20:4, 2018.

[7] Bonomi, Flavio and Milito, Rodolfo and Zhu, Jiang and Addepalli, Sateesh. Fog computing and its role in the internet of things. *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, 13–16, 2012.

[8] Verbelen, Tim and Simoens, Pieter and De Turck, Filip and Dhoedt, Bart. Cloudlets: Bringing the cloud to the mobile user. *Proceedings of the third ACM workshop on Mobile cloud computing and services*, 29–36, 2012.

[9] Famaey, Jeroen and Latré, Steven and Strassner, John and De Turck, Filip. A hierarchical approach to autonomic network management. *2010 IEEE/IFIP Network Operations and Management Symposium Workshops*, 225–232, 2010.

[10] Deboosere, Lien and Vankeirsbilck, Bert and Simoens, Pieter and De Turck, Filip and Dhoedt, Bart and Demeester, Piet. Efficient resource management for virtual desktop cloud computing. *The Journal of Supercomputing, Springer*, 62:741–767, 2012.

[11] Pradhan, Pandaba and Behera, Prafulla Ku and Ray, BNB. Modified round robin algorithm for resource allocation in cloud computing. *Procedia Computer Science, Elsevier*, 85:878–890, 2016.

[12] Moens, Hendrik and De Turck, Filip. VNF-P: A model for efficient placement of virtualized network functions. *10th International Conference on Network and Service Management (CNSM) and Workshop*, 418–423, 2014.

[13] Bhamare, Deval and Samaka, Mohammed and Erbad, Aiman and Jain, Raj and Gupta, Lav and Chan, H Anthony. Optimal virtual network function placement in multi-cloud service function chaining architecture. *Computer Communications, Elsevier*, 102:1–16, 2017.

[14] Nadareishvili, Irakli and Mitra, Ronnie and McLarty, Matt and Amundsen, Mike. Microservice architecture: aligning principles, practices, and culture. *" O'Reilly Media, Inc."*, 2016.

[15] Santos, José and Wauters, Tim and Volckaert, Bruno and De Turck, Filip. Towards delay-aware container-based Service Function Chaining in Fog Computing. *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium. Accepted for publication*, 2020.

[16] Mnih, Volodymyr and Badia, Adria Puigdomenech and Mirza, Mehdi and Graves, Alex and Lillicrap, Timothy and Harley, Tim and Silver, David

and Kavukcuoglu, Koray. Asynchronous methods for deep reinforcement learning. *International conference on machine learning*, 1928–1937, 2016.

[17] Van Hasselt, Hado and Guez, Arthur and Silver, David. Asynchronous methods for deep reinforcement learning. *Thirtieth AAAI conference on artificial intelligence*, 2016.

[18] Hessel, Matteo and Modayil, Joseph and Van Hasselt, Hado and Schaul, Tom and Ostrovski, Georg and Dabney, Will and Horgan, Dan and Piot, Bilal and Azar, Mohammad and Silver, David. Rainbow: Combining improvements in deep reinforcement learning. *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[19] Arkian, Hamid Reza and Diyanat, Abolfazl and Pourkhalili, Atefe. MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications. *Journal of Network and Computer Applications*, 82:152–165, 2017.

[20] Brogi, Antonio and Forti, Stefano. QoS-aware deployment of IoT applications through the fog. *IEEE Internet of Things Journal*, 4:1185–1192, 2017.

[21] Skarlat, Olena and Nardelli, Matteo and Schulte, Stefan and Dustdar, Schahram. Towards qos-aware fog service placement. *2017 IEEE 1st international conference on Fog and Edge Computing (ICFEC)*, 89–96, 2017.

[22] Yasmeen, Anila and Javaid, Nadeem and Rehman, Obaid Ur and Iftikhar, Hina and Malik, Muhammad Faizan and Muhammad, Fatima J. Efficient resource provisioning for smart buildings utilizing fog and cloud based en-

vironment. *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*, 811–816, 2018.

[23] Yao, Jingjing and Ansari, Nirwan. Fog Resource Provisioning in Reliability-Aware IoT Networks. *IEEE Internet of Things Journal*, 2019.

[24] Chiti, Francesco and Fantacci, Romano and Paganelli, Federica and Picano, Benedetta. MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications. *IEEE Transactions on Network and Service Management*, 16:980–989, 2019.

[25] Goethals, Tom and De Turck, Filip and Volckaert, Bruno. FLEDGE: Kubernetes compatible container orchestration on low-resource edge devices. *SC2 2019, the 9th International Symposium on Cloud and Service Computing*, 1–16, 2019.

[26] Goethals, Tom and De Turck, Filip and Volckaert, Bruno. Adaptive fog service placement for real-time topology changes in Kubernetes clusters. *CLOSER2020, the 10th International Conference on Cloud Computing and Services Science*, 161–170, 2020.

[27] Scalingi, Alessio and Esposito, Flavio and Muhammad, Waqar and Pescapé, Antonio. Scalable provisioning of virtual network functions via supervised learning. *2019 IEEE Conference on Network Softwarization (NetSoft)*, 423–431, 2019.

[28] Subramanya, Tejas and Harutyunyan, Davit and Riggio, Roberto. Machine learning-driven service function chain placement and scaling in MEC-enabled 5G networks. *Computer Networks*, 166:106980, 2020.

[29] Kim, Hee-Gon and Lee, Do-Young and Jeong, Se-Yeon and Choi, Heeyoul and Yoo, Jae-Hyung and Hong, James Won-Ki. Machine Learning-Based Method for Prediction of Virtual Network Function Resource Demands. *2019 IEEE Conference on Network Softwarization (NetSoft)*, 405–413, 2019.

[30] Kimovski, Dragi and Ijaz, Humaira and Saurabh, Nishant and Prodan, Radu. Adaptive nature-inspired fog architecture. *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*, 1–8, 2018.

[31] Mao, Hongzi and Alizadeh, Mohammad and Menache, Ishai and Kandula, Srikanth. Resource management with deep reinforcement learning. *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, 50–56, 2016.

[32] Gai, Keke and Qiu, Meikang. Optimal resource allocation using reinforcement learning for IoT content-centric services. *Applied Soft Computing*, 70: 12–21, 2018.

[33] Li, Ji and Gao, Hui and Lv, Tiejun and Lu, Yueming. Deep reinforcement learning based computation offloading and resource allocation for MEC. *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, 1–6, 2018.

[34] Troia, Sebastian and Alvizu, Rodolfo and Maier, Guido. Reinforcement Learning for Service Function Chain Reconfiguration in NFV-SDN Metro-Core Optical Networks. *IEEE Access*, 7:167944–167957, 2019.

[35] Chen, Xiaosha and Leng, Supeng and Zhang, Ke and Xiong, Kai. A

machine-learning based time constrained resource allocation scheme for vehicular fog computing. *China Communications*, 16:29–41, 2019.

[36] Santos, José and Wauters, Tim and Volckaert, Bruno and De Turck, Filip. Towards End-to-End resource provisioning in Fog Computing over Low Power Wide Area Networks. *Submitted to Journal of Network and Computer Applications*, 2020.

[37] Watkins, Christopher JCH and Dayan, Peter. Q-learning. *Machine learning*, 8:279–292, 1992.

[38] Brockman, Greg and Cheung, Vicki and Pettersson, Ludwig and Schneider, Jonas and Schulman, John and Tang, Jie and Zaremba, Wojciech. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Communication Networks and Service Management in the Era of Artificial Intelligence and Machine Learning

Chapter 7

Reinforcement Learning for Service Function Chain Allocation in Fog Computing

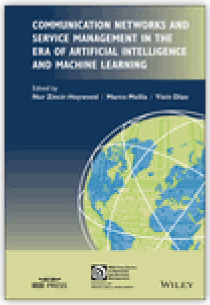
José Santos, Tim Wauters, Bruno Volckaert, Filip De Turck

Book Editor(s):Nur Zincir-Heywood, Marco Mellia, Yixin Diao

First published: 03 September 2021
https://doi.org/10.1002/9781119675525.ch7

Summary

Recently, distributed cloud infrastructures have become a potential business opportunity for most service providers due to the exponential growth of connected devices. The advent of the Internet of Things (IoT) and softwarized networks made centralized cloud systems impractical. In response, Fog Computing (FC) emerged, enabling the deployment of services on computational resources from the cloud up to the edge. However, the adoption of FC concepts is still in its early stages and challenges persist to fully benefit from fog–cloud infrastructures. One of them is known as Service Function Chaining (SFC) where providers benefit from network softwarization to create virtual chains of connected services. Recent research has tackled SFC allocation through theoretical modeling and heuristic algorithms, which often cannot cope with the dynamic behavior of the network. Thus, in this chapter, we explore a subset of machine learning (ML) called Reinforcement Learning (RL) to provide an efficient solution for SFC allocation in FC. The proposed approach learns about the best resource allocation decisions, focused on energy efficiency from a previously presented mixed-integer linear programming (MILP) formulation. Results showed that RL algorithms perform comparably to state-of-the-art ILP-based implementations while offering more scalable solutions. Future research directions and open challenges are discussed.



Communication Networks and Service Management in the Era of Artificial Intelligence and Machine Learning

Editor(s): Nur Zincir-Heywood, Marco Mellia, Yixin Diao
First published: 27 September 2021
Print ISBN: 9781119675501 | Online ISBN: 9781119675525 | DOI: 10.1002/9781119675525

About this book

COMMUNICATION NETWORKS AND SERVICE MANAGEMENT IN THE ERA OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Discover the impact that new technologies are having on communication systems with this up-to-date and one-stop resource

...

Table of Contents

GO TO PART

Export Citation(s)

Training and Support
DMCA & Reporting Piracy

Opportunities

Subscription Agents
Advertisers & Corporate Partners

Connect with Wiley

The Wiley Network
Wiley Press Room

WILEY