An Automated VNF Manager based on Parameterized-Action MDP and Reinforcement Learning

by

Xinrui Li

Thesis submitted to the University of Ottawa
In partial fulfillment of the requirements
For the M.A.Sc. degree in
Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Managing and orchestrating the behaviour of virtualized Network Functions (VNFs) remains a major challenge due to their heterogeneity and the ever increasing resource demands of the served flows. In this thesis, we propose a novel VNF manager (VNFM) that employs a parameterized actions-based reinforcement learning mechanism to simultaneously decide on the optimal VNF management action (e.g., migration, scaling, termination or rebooting) and the action's corresponding configuration parameters (e.g., migration location or amount of resources needed for scaling). More precisely, we first propose a novel parameterized-action Markov decision process (PAMDP) model to accurately describe each VNF, instances of its components and their communication as well as the set of permissible management actions by the VNFM and the rewards of realizing these actions. The use of parameterized actions allows us to rigorously represent the functionalities of the VNFM in order perform various Lifecycle management (LCM) operations on the VNFs. Next, we propose a two-stage reinforcement learning (RL) scheme that alternates between learning an action-value function for the discrete LCM actions and updating the actions parameters selection policy. In contrast to existing machine learning schemes, the proposed work uniquely provides a holistic management platform the unifies individual efforts targeting individual LCM functions such as VNF placement and scaling. Performance evaluation results demonstrate the efficiency of the proposed VNFM in maintaining the required performance level of the VNF while optimizing its resource configurations.

Acknowledgements

First of all, I would like to express the very appreciation to my parents for their support from the very beginning and their cares when I was promoting this degree. Without their support I can absolutely say that I cannot keep going when I felt bad.

Also, I would like to express the very appreciation to Dr. Samaan and Dr. Karmouch, my supervisors, for their patient instructions. Not only from the study I received so much help but also I was warmed by their cares when I had troubles. It is their encouragement that helps me finish this thesis and I will never forget that.

Lastly, I would like to express the very appreciation to the partner in our lab, Zhuonan. Within a year, I saw her success from when she was new to the environment to the researcher who finished her project so perfectly. We fixed lots of problems one by one together and encourage each other at the most difficult times.

Table of Contents

List of Tables

List of Figures

Nomenclature

Abbreviations

A2C Advantage actor-critic

ADP Approximate Dynamic Programming

AI Artificial Intelligence

CMDP Constrained Markov Decision Process

CP Connection Point

CPU Central Processing Unit

DDPG Deep Deterministic Policy Gradient

DECRL Decentralized Reinforcement Learning

DL Deep Learning

DNN Deep Neural Network

DQN Deep Q-Network

DRL Deep Reinforcement Learning

ICN Information-Centric Network

LSTM Long Short-term Memory

MANO Management and Orchestration

MC Monte Carlo

MCC Mobile Cloud Computing

MDP Markov Decision Process

NF Network Function

NFV Network Function Virtualisation

NFVI Network Function Virtualisation Infrastructure

NFVO Network Function Virtualisation Orchestrator

NN Neural Network

NS Network Service

PI Policy Iteration

QoE Quality of Experience

QoS Quality of Service

RAN Radio Access Network

RL Reinforcement Learning

RNN Recurrent Neural Network

SDN Software-Defined Network

TD Temporal Difference

VDU Virtual Device Unit

VI Value Iteration

VIM Virtualised Infrastructure Manager

VL Virtual Link

VM Virtual Machine

VNF Virtualised Network Function

VNFC Virtualised Network Function Component

VNFM Virtualised Network Function Manager

WAN Wide Area Network

Chapter 1

Introduction

In the recent 30 years, networks have expanded to an enormous scale. In the early years, networks were designed to provide connectivity, i.e., to enable packets sent from one host to arrive at the desired destination. Routing techniques, switching, and flow scheduling are examples of core functions. However, other network functionalities such as network security, caching, and encoding was implemented by end-users. This design resulted in great success for network application development and evolution without requiring upgrades to the network itself. This development model was successful for quite a long time, leading to a large range of software and devices connecting to networks today. As a result, network operators had to implement additional network functionalities to support the upgrades of network applications. Today's networks include lots of equipment such as routers, switches, and middleboxes [5]. Proxies, firewalls, network address translators, intrusion detection systems, load balancers, flow monitor, and WAN optimizers are specific services that network equipment provide. The number of middleboxes in different types of networks is comparable with the number of switches and routers. The number of deployed middleboxes grows steadily. It is shown in a recent survey that the cost of middleboxes accounts for over two-fifths of the network hardware in enterprise networks [6]. The mis-configuration is responsible for about 40% of high-severity datacenter failures. Middleboxes have several drawbacks:

- It is hard to re-deploy and re-configure hardware middleboxes once the physical

network is constructed. This will affect the rate where networks can offer new network services.

- Middleboxes typically implement a fixed number of services and handle a fixed load. This leads to redundant operations for network operators because when there are device failures and increased demand, operators need to re-install several instances.

- Middleboxes need frequent replacement due to device damages as well as protocol updates (e.g., updates in LTE standards)

Typically, an NF refers to one or a set of specialized hardware devices. With virtualization techniques, Network Function Virtualisation (NFV) is promoted. NFV focuses on changing the way network operators design, deploy and manage the network infrastructures. The goal of NFVs is to remove the dedicated expensive middleboxes and the massive amount of specialized hardware devices and deploy network services (NSs) and virtualized network function (VNFs) on top of industry-standard servers or containers. In NFV, NFs are deployed as virtual machines (VMs) running on commodity computing units, ensuring a rapid NF deployment within a cloud platform, dynamic response to updates in network load and failures. It can also update the protocol and application software in a cheaper and safer mode.

1.1 Motivation

Recently, the premise of NFV [7] has received immense attention from researchers in both academia and industry. This can be attributed to its promise of cost reduction, and ease of manageability as NFV replaces hardware middleboxes with commodity servers hosting VNF instances. This approach allows operators to dynamically build up and tear down NS that is comprised of customized VNFs. For example, with the idea of NFV, a network operator is likely to run a software-based firewall on an x86 platform via virtualization techniques. NFV provides possibilities for legacy networks. For example, NFV makes it possible to decouple software from hardware and allows to separate NS development

and maintenance for both software and hardware. On the other hand, it offers a more flexible network function deployment since the network operator is able to re-configure the network function to another cloud cluster in the same data center in terms of needs. Software-defined network (SDN) is another hot network virtualization topic in industry and academia. SDN is promoted to simplify the network management and make the deployment of NS easier by decoupling the control plane and the data plane. A service function chain (SFC) [8, 9] is composed of a list of required network functions and the corresponding order that has to be applied to the packet belonging to the specific data flow. In NFV, a NS is composed of one or several VNFs in an ordered sequence. For example, an IDS may need to inspect the packet before the data compression and encryption. Meanwhile, in terms of different requirements from customers, the sequence of network functions can be different. Figure 1.1 shows an example of an SFC with five different network functions.

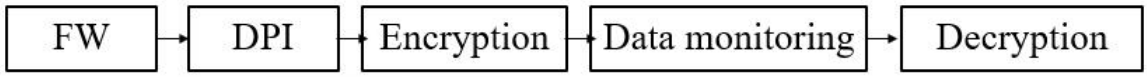

Figure 1.1: Example of a SFC/NS

NFV is promoted with unique architecture. With the promise of capital expenditures and operating expenses (CAPEX/OPEX) [10, 11] savings, network operators have invested in developing a novel European Telecommunications Standards Institute (ETSI) standardized VNF framework [12]. The architecture is comprised of two layers, the VNF infrastructure (VNFI) of compute, storage, and network resources, that is hosting the second layer of VNFs using different means of visualization such as VMs or containers. The management and orchestration (MANO) entity in the ETSI architecture maintains catalogs of offered NSs, their SFC compositions, and the available flavors of the VNFs. To manage the two layers, MANO is comprised of three main components: the NFV orchestrator (NFVO [13], the VNF manager (VNFM) [14], and the virtual infrastructure manager (VIM) [15]. These components interact together and with the physical and VNF layers to manage the deployment and lifecycle of the offered NSs, their SFCs, and the physical and virtual resources.

One of the functionalities of MANO is to fulfill the contracted Service Level Agreement (SLA) [16] requirements of the hosted NSs [17] while optimizing the number of consumed resources on the VNFI. Examples of these requirements include the experienced latency by the served flows, their service rate, service availability, and cost [18]. Once these requirements are mapped to service requirements for each VNF instance in the NS, the VNFM is tasked with performing various lifecycle management (LCM) operations (e.g., migration, scaling, termination, and rebooting) in various components of the VNF to maintain its assigned SLA defined performance values [18].

The problem can be stated as follows. The central problem of an NFV-based platform is that it lacks an efficient-enough mechanism to execute the desired operations defined in the ETSI-NFV standards [19]. Existing research efforts have addressed several critical issues for VNF management, including placement, resource allocation, scaling, and migration [20]. Notably, machine learning (ML) based schemes have been shown to provide a promising solution to realize these functionalities (e.g., VNF scaling [21]). Nonetheless, the majority of these efforts do not provide a holistic approach that can aid the VNFM in selecting, at any given point in time, nor given a measured behavior of the VNF, the most appropriate VNF reconfiguration action among all possible LCM operations [22]. Furthermore, most VNFs are VM-based, which wastes a lot of computing resources on OS instantiating in each VM instance.

We envision that to simplify the management and reduce the complexity of the VNF LCM problem and guarantee the performance of the VNF, and the VNFM should be able to either proactively or reactively respond to VNF status updates and to perform proper LCM operations. Given this, the objective of the thesis is to improve the efficiency of the VNFM to make better decisions on VNF LCM problems and resource allocation optimizations on the basis of the long-term performance of NSs.

1.2 Contribution

In this thesis, we aim at achieving full automation of the functionality of the VNFM. We realize this goal by first developing a novel model to fully describe the VNFs, and the VNFM behavior using a parameterized-action-based Markov decision process (PAMDP) [23]. In contrast to existing approaches, the model can incorporate different actions of the VNFM (e.g., scaling, migration, termination, and rebooting of various VNF components) as well as the needed configuration parameters for these actions. We then develop a two-stage reinforcement learning mechanism to efficiently learn optimal policies of selecting the appropriate LCM action-parameters pairs. In the first stage of the algorithm, a discrete LCM action is selected using value-functions. In the second stage, the action configuration parameters are repetitively refined using policy gradient methods. Once optimal policies are learned, they are employed to fine-tune the performance of the VNF based on monitored performance measures and the SLA-based desired behavior. We focus on the LCM of containerized VNF on Kubernetes, an open-source container-orchestration system for automating computer application deployment, including scaling operations, migrations, and renewing. To execute a scaling operation for VNFs, we need to investigate the number of requested resources in the VNF deployment file. The same applies to other VNF LCM operations, such as migration and renewing operations. In comparison, migration operations need a global view of the current SFC and global resource allocation. In order to solve these problems, we proposed an intelligent VNF manager with RL-based agents inside, which can automatically decide which action to take and the detailed resources we need to assign to the VNF instances. More precisely, we study the relationship between VNFs as well as the correlations between different VNF LCM operations and learn the RL agent to fulfill the SLA of VNFs and NS. We can summarize the contributions into the following:

- We formulate the VNF LCM problem by constructing a Parameterized Action Markov Decision Process (PAMDP). In the model, VNFs are parted into different components. In the model, the processed unit is called virtual device unit (VDU) which is represented by containers. We formulate the PAMDP problem with a parameterized

action space and multiple dimensions of states space and apply different RL algorithms. In our model, the action is determined not only via a discrete manner, but also adjusted by the action-parameter.

- We design a compound parameterized action RL algorihtm where we apply Q-learning on the discrete action selection and Policy Gradient on action parameter selection. We demonstrate through performance evaluation results by comparing to the plain VNF LCM operations.

1.2.1 Publication

Xinrui Li, Nancy Samaan and Ahmed Karmouch. An Automated VNF Manager based on Parameterized-Action MDP and Reinforcement Learning. In *2021 IEEE International Conference on Communications(ICC), 2021.*

1.3 Thesis outline

The reminder of this thesis is organized as follows.

- Chapter 2 gives a background of NFV and discusses the VNF LCM problems with typical relevant solutions. Then we introduce the fundamentals of RL and related RL algorithms.
- Chapter 3 presents the system model and architecture, and formulates the VNF LCM problem as a Parameterized Action Markov Decision Process problem. Then we present our parametrized-based RL algorithm
- Chapter 4 presents performance evaluation results.
- Chapter 5 concludes the thesis and discusses possible future work

Chapter 2

Background and Literature Review

This chapter provides a literature review on the NFV development process as well as VNF LCM and orchestration problem solutions. Section 2.1 gives a basic background of NFV development. Section 2.2 is a more detailed introduction of NFV architecture, including all the NFV components and their functionalities. Section 2.3 presents a fundamental introduction of virtualization techniques such as virtual machines and containers. Section 2.4 gives examples of current VNF LCM solutions and discusses the drawbacks. Section 2.5 is an introduction for RL and talks about current RL solutions. Section 2.6 gives a summary of this chapter.

2.1 Background of Network Function Virtualization

NFV is promoted with the development of the virtualization technology [24]. Unlike traditional network functions, the dedicated hardware is removed and is replaced with the available software running on commercial off-the-shelf (COTS) [25] servers and works on top of VMs and containers. NFV has raised immense attention from researchers in academia and industry. VNF is a core functional block that runs different network functions with various configuration options. Diverse combinations of VNFs form SFC to fulfill SLA from customers.

2.1.1 History of NFV evolution

Most network equipment is in fixed and dedicated architecture, and network functions execute their job on the standard vendor-provided hardware and software. This fact restricts the service provider and the network managers and operators, for its dependency between the two components is strongly bound together. As the network service becomes more and more complicated, telecommunication service providers and operators are likely to be asked to deploy and install new dedicated hardware appliances. At the same time, they have to make space for future deployment and consider power consumption problems in their deployed place. All of these lead to CAPEX and OPEX losses [26], while increasing the difficulties of platform management since the hardware is definitely becoming more complex.

NFV is able to decouple the dedicated hardware of network service from its software. It targets the development of network service by applying multiple standard hardware from various vendors that can cooperate with multi-vendor software platforms. As a result, NFV will change the current coupling problem between the dedicated hardware and its software, and also, it can be expected that CAPEX and OPEX will be reduced if we apply such an architecture to current network services. NFV-based network services run in edge cloud and in data centers in terms of deployment requirements to respond to geographically dispersed deployment requests.

NFV is developed from the virtualization techniques, which enable customers to share the physical resources and functionalities through virtual machines and containers. Through network virtualization, VNFs share the same network element hardware resources, leading to decoupling between network services and its hardware. Multiple network services can be re-deployed on the same network element based on this fact. Similarly, for higher flexibility, NFV can migrate network functions to a virtual cloud-based infrastructure.

2.2 NFV Architecture

NFV is promoted by the European Telecommunications Standards Institute (ETSI)[12]. ETSI defines the NFV architectural framework, and VNFs are deployed. They are working on a bottom layer called network function virtualization infrastructure (NFVI), which is composed of commodity servers that has virtualization eligibility and can be fully abstracted and logically partitioned. Above the NFVI layer, there are VNFs running in the middle layer; more precisely, one VNF is typically mapped to one or multiple VM or containers in the NFVI. One or several VNFs can form up a complete NS or an SFC in the term of SDN. NS, VNF, and their corresponded NFVI are connected and steered by a management and orchestration system(MANO) [19]. The NFV-MANO [27] is the brain of the NFV entity and is driven by the metadata predefined in the characteristics of the NS and their inner VNFs. To well manage and orchestrate the NFV entity, the MANO panel has three core components corresponding to the NS, VNF, and NFVI. Virtualized infrastructure manager (VIM) is at the bottom of the MANO panel, and the main functionality is to collect the executing data from the NFVI, as well as to manage the NS and its VNF in the most direct manner. The VNFM is the core component for VNF instance management, also in charge of the coordination between the network function virtualization orchestrator (NFVO) and VIM. On top of MANO is the NFVO, the main controller for NS, which is in charge of the global orchestration tasks. NFVO keeps the metadata of the whole map of NFV instances and updates the policy to the bottom MANO entity such as VNFM and NFVI when detecting requests update from the Office of Strategic Services (OSS) [28] and business support system (BSS) [29] as well as self-needed updated from VNFM and NFVI. The MANO panel and NFV instances form up the NFV entity as it is shown in Figure 2.1.

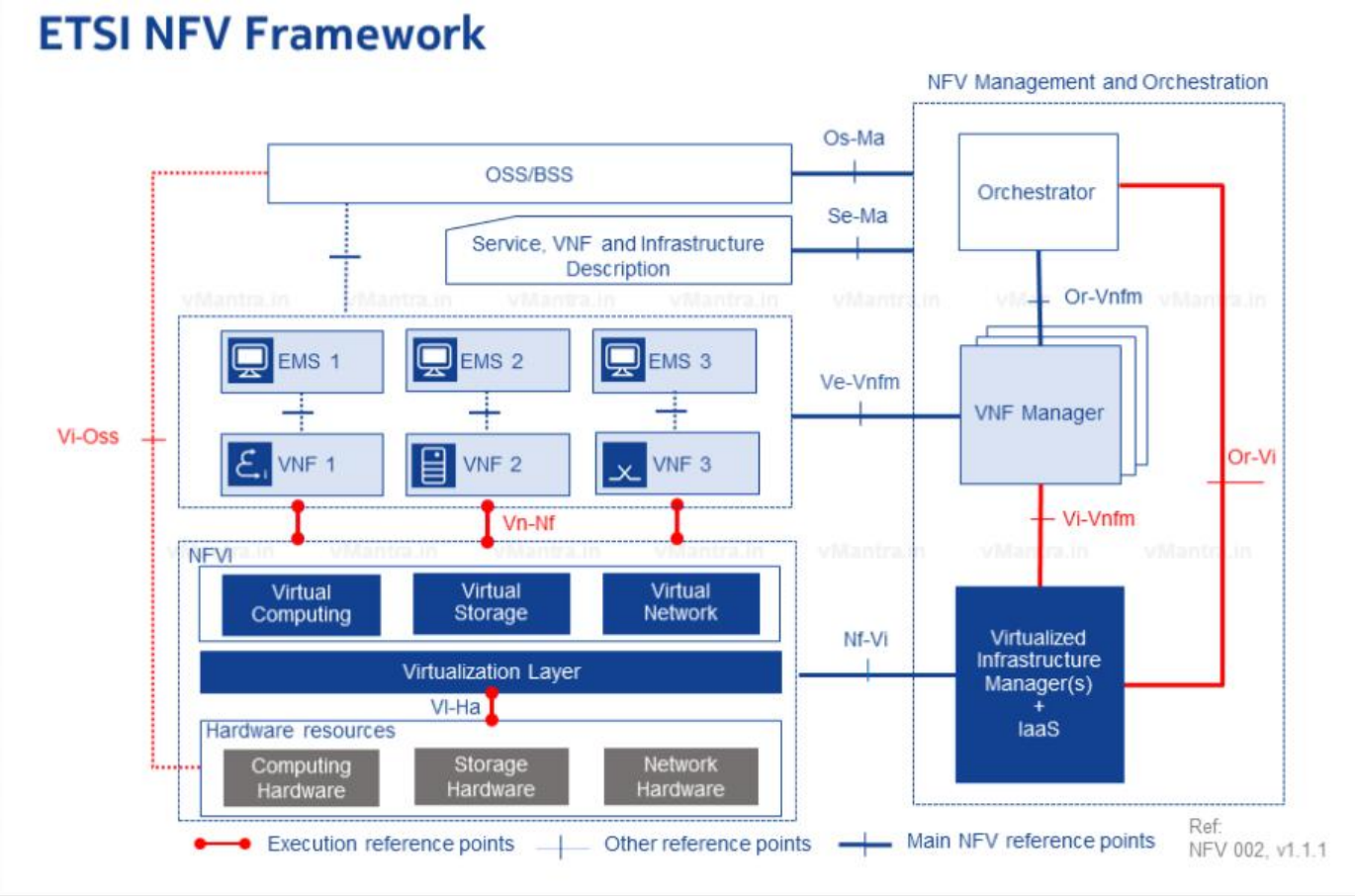

Figure 2.1: Architecture of ETSI-NFV.[1]

2.2.1 NFV Orchestration

We introduce the core component of the NFV-MANO framework and the NFV orchestrator [30]. The NFV orchestrator is the main functional block of the MANO architecture, which is responsible for the management and orchestration of NFVI resources across multiple VIMs as well as the global management on the VNF layer on top of them, and of the LCM of the network services.

Two main responsibilities of NFVO are defined by ETSI where it requires the NFVO must fulfill the management, and orchestration aspects are summarized below:

- Resource orchestration. The resource orchestration is a function block in NFVO which satisfies functions that handle the resource release and allocation from NFVI (i.e., release computation, storage, and network resources from NFVI).
- Network service orchestration. The network service orchestration is a function block in NFVO, which satisfies MANO function requirements by handling the life-cycle management, including onboarding, instantiation, scaling, instance-updating, and termination of the network services as well as the other correlated operations with VNF forwarding graphs (VNFFG) [1]. The NFVO manages the coordinated groups of VNF instances through network service orchestration functions to provide well-managed complex network services. It manages the joint configuration and instantiation. Meanwhile, it has a global view of the required connection between VNFs and should be responsible for dynamic VNF configuration update for further use cases (e.g., NFVO is able to push the scaling policy to VNFM for high availability concerned in NSs)

Network service

In general, an NS can be considered as a forwarding graph of network functions (NFs), which are interconnected by the below network infrastructure. Network functions can be implemented in a single operator network or multiple ones interconnected between different

operator networks. The underlying network function behavior contributes to the behavior of the high-level service. As a result, the network service behavior is a compound one combining the behavior of its constituent functional blocks, which includes individual NFs, NF sets NF forwarding graphs, and the infrastructure network.

The endpoints and the NFs in the network service are represented as nodes and correspond to devices, applications, and physical server applications. An NF forwarding graph is composed of network function nodes connected by logical links, which can be either unidirectional or bidirectional, in some cases can be multicast or broadcast. To create an end-to-end network service, a simple example is using the forwarding graph to represent a chain of network function, which can also be represented as an SFC. An example of such an end-to-end network service can include a smartphone, a wireless network, a FW, a LB, and a set of CDN servers. The NFV area of activity is within the operator-owned resources. Hence, a customer-owned device (i.e., a mobile phone outside the scope of an operator) cannot exercise its authority on it. On the other hand, virtualization and network-hosting of customer functions is possible and is in the scope of NFV [31]

Figure 2.2 represents an end-to-end network service managed by the NFVO. It also includes a second nested NF forwarding graph as indicated by the network function block nodes in the middle of the figure, which are also interconnected by some logical links. As shown in the figure, the endpoint is linked to NFs via NFVIs, leading to a logical interface between the endpoint and a network function. These logical interfaces are represented with dotted lines in the figure. The outer end-to-end NS is composed of endpoint A, the inner NF forwarding graph and the endpoint B, where the internal NF forwarding graph is comprised of $NFs(NF1, NF2, NF3)$. Those NFs are interconnected by logical links provided by the $Infrastructure\ network2$.

Figure 2.3 provides an example of an end-to-end NS representing different layers included for its virtualization process. In the example, an an-end-to-end network service can be represented by only VNFs and the endpoints. By decoupling, the hardware and software in NFV are realized by a virtualization layer (hypervisor layer) where it abstracts the hardware resources of the NFV infrastructure. The NFVI-PoPs in Figure 2.3 include

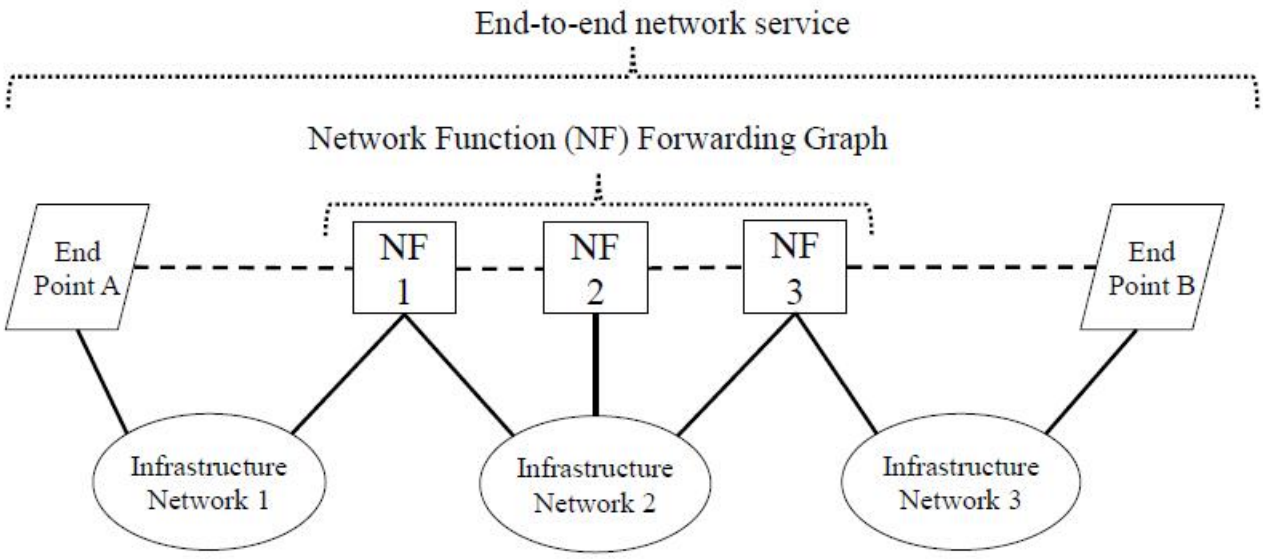

Figure 2.2: Representation of an end-to-end network service[1]

resources for computation, storage, and networking deployed by a network operator.

VNFs run on top of the virtualization layer, which can be considered as part of NFVI, indicated by the arrow with the label "virtualization" in Figure 2.3. The VNF forwarding graph (VNFFG) corresponding to the NF forwarding graph is also presented in Figure 2.2. As shown in the figure, it depicts the case of a nested VNFFG (i.e., VNFFG2), which is constructed from other VNFs(i.e., VNF-2A,VNF-2B, and VNF-2C). The objective is to determine the interfaces between NFs and VNFs, and the infrastructure in a multi-vendor environment is likely to be based upon accepted standards(i.e., standardized by an SDO or an open de-facto standard) The NFVO applies the resource orchestration functionality to abstract the access to the available resources provided by the NFVI beneath to services. Features of NFVO can be summarized as follows:

- Authorization and validation of NFVI resources requests from the VNF manager. This ensures a control process on the allocation of requested resources interacting within one NFVI-PoP or across multiple NFVI-PoPs;
- Resource management in NFVI, which includes the basic resource distribution, reservation, and allocation of NFVI resource to NS and VNF instances; available resource can be either obtained from existed NFVI resources or from multi-vendor enabled

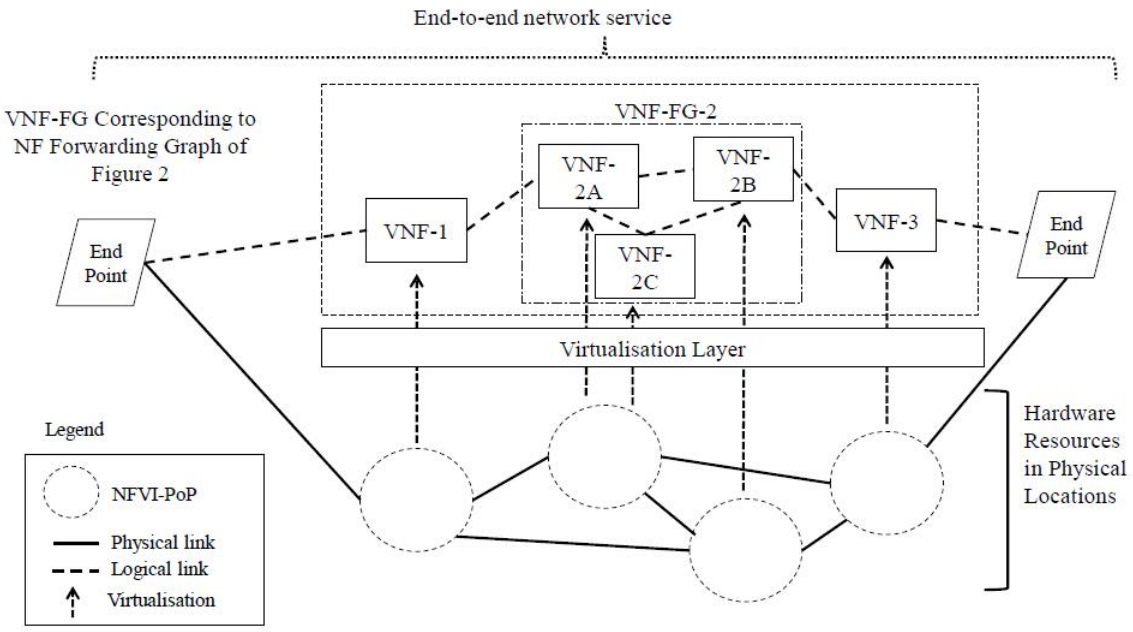

Figure 2.3: Example of an end-to-end network service with VNFs and nested forwarding graphs[1]

NFV infrastructure. The NFVO resolves the VIM placement locations as well to provide an acceptable resource allocation mechanism under the requirement of BSS/OSS and SLA.

- Management of the corresponding interfaces between VNF layer and NFVI layer. Precisely speaking, the underlying VNF instance and the NFVI resource are allocated to it. NFVO should use the resource repositories and information received from the VIMs.

- Policy management and enforcement, NFVO should implement policies on NFVI resources, including a series of operations such as access control, preservation, and optimization of the placement based on geographical, affinity, and/or regulatory rules.

- Collection of information regarding the use cases by single or multiple VNFs of NFVI resources.

Besides, network service orchestration (NSO) in the NFVO is responsible for the management of the services exposed by the VNF manager (VNFM) and by the resource orches-

tration functional block (RO). Generally, the NSO is exposed via interfaces consumed by other NFV-MANO framework or external entities:

- Management of NS deployment configurations and VNF packages. Those configurations will be verified by the NFVO to ensure integrity, authenticity, and consistency, and NFVO will store the images provided in VNF packages via available NFVI-PoPs.
- NS life-cycle management, including NS instantiation, updating, querying, scaling and termination. Meanwhile, it also requires a collection of work for performance evaluation and events recording.
- Direct management of VNFM.
- Management of VNFs from a higher level, the VNFM is responsible for reporting the VNF-LCM process to the NSO.
- Management of NFVI resources from a higher level, the VIM should work in coordination with the NSO to control the workflow of the current NS for validation and authorization.
- Management of the VNFFG where the logical end-to-end NS instance is defined.
- Automation of NS instance management by triggering automatic operational management actions for NSs and VNFs. This process follows the principle defined in the NS configuration templates.
- Policy management and evaluation for the NSs and VNFs. NSO is responsible for policies generating concerning scaling, fault, geography, affinity/anti-affinity regulatory rules, and NS catalogs.
- Management of the NS instance through their lifecycles to guarantee the visibility and integrity. NFVO is also responsible for a proper relationship between a NS and a VNF.

2.2.2 VNF Manager

The VNF manager[32] is an NFV-MANO function block that satisfies the MANO aspects of VNFs within the LCM of the VNF instances. A VNF manager is responsible for one more multiple VNFs in terms of the SFC. No matter what type of VNFs it handles, the VNFM must have the support functions that support the associated VNFs. Similar to NFVO, the VNFM exposes its functionalities to other MANO entities with defined interfaces:

- VNF configuration and instantiation with a VNF deployment template
- Investigation of the feasibility of a VNF instantiation process.
- VNF instance software update.
- VNF instance online/offline modification.
- VNF instance scaling operation.
- VNF instance termination
- VNF instance resiliency concern. (Auto healing, fault management)
- Handle VNF notifications in the VNF lifecycle when there are changes.
- Handle VNF verification and integrity update in the lifecycle of a VNF instance.
- Information consolidation from NFVI, including performance measurement results, faults, and events corresponded to the VNF. Coordinate and handle the configuration report sent from VIM and EM.

VNFM should be responsible for all VNF instances. Each VNF instance is defined in the VNFM with a specific deployment template: virtualized network function descriptor(VNFD), which is stored in a VNF catalog corresponding to a VNF package. A VNFD is a description file that defines the operational behavior of a VNF and specifies the deployment details of a VNF with a full map of attributes and requirements. In general,

NFVO applies copies of VNFD and correlated VNFFG to create a network service descriptor (NSD). And this is how the two functional blocks bind with each other in terms of VNF instances. MANO entity uses VNFDs to instantiate VNFs, manage the lifecycles, and coordinate with NFVI for resource concerns.

Since the VNFM has direct access to the repository of available VNF packages, the VNFM should list all the possible packages inside the VNFD to ensure a different version of VNF implementations of the same functionality on other service provider environments. Figure 2.4 shows the relationship between NFV-MANO and the VNF instances and their descriptors. As shown in the figure, the configuration related to the management and

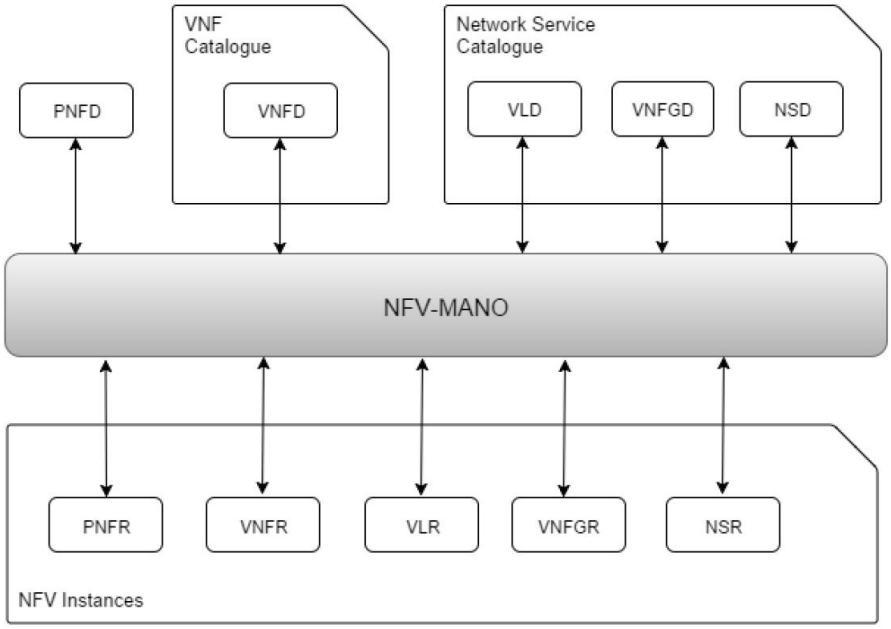

Figure 2.4: Relationship between MANO entity and VNF instances and descriptors[2]

orchestration operations is kept in several repositories:

- NS catalogue. NS catalog contains a full map of the on-boarded network service. As it is designed to support the logic creation and management of the deployment templates of a NS. In this case, the network service descriptor has similar functionalities but is kept in an authentic way. The same applies to virtual link descriptors (VLDs) and VNF forwarding graph descriptor (VNFFGD)

- VNF catalogue. VNF catalog is a repository for all on-boarded VNF packages. Similarly, the VNF catalog is designed to support the logic creation and management of the deployment templates (VNFD) through exposed interface operations from NFVO. The VNFM and NFVO have the absolute query authorization to search, retrieve and validate VNFDs.

- NFV instance repository. NFV instance catalog is a functional logic block that holds information about NS and VNF instances, which is represented by their respective VNF records (VNFRs) and NS records (NSRs). These records are updated within the VNF-LCM and NS-LCM, reflecting the changes resulting from the executing of LCM operations.

- NFVI resource repository. The NFVI resource repository collects information about the currently available and reserved NFVI resources, which the VIMs abstract. The repository has excellent values in resource allocation, reservation concerns. As a result, this component owns a great place in the NFVO's resource orchestration and governance roles via available tracking on NFVI allocated and reserved resources against the initial associated NS and VNF instances.

VNF Lifecycle Management

To manage the VNF properly and efficiently, a VNFM should include traditional fault management configuration management, performance management and security management(FCAPS) as well as the lifecycle management[22] including operations such as:

- instantiate VNF, to create a VNF using the VNF on-boarding artifacts

- scale VNF, to increase or decrease the capacity of the VNF

- upload and upgrade VNF, to configure the VNF software when VNF update is available

- terminate VNF, to release the associated NFVI resources and return to the resource pool

- migrate VNF, to migrate VNF to another NFVI

A VNF instance is deployed with a template that contains the deployment and operational behavior requirements and is stored during the VNF onboarding process. The deployment template describes the attributes and requirements necessary to instantiate such a VNF and abstractedly captured to manage its lifecycle. During the lifecycle of a VNF, the VNFM pushes the monitor command to VIM to monitor the key performance indicators (KPIs) of a VNF. If such KPIs are captured in the deployment template, the information is likely to be used for other LCM operations like scaling operations. Scaling operations include changing the configurations of the virtualized resources, either adding up or cutting down the existing size of resources such as virtualized CPU numbers and memory. Besides, configuration changes also correspond to virtualized instances number changes, leading to scaling in and scaling out operations. A VNF is composed of a VNF component (VNFC)[33], and a VNFC has one or several corresponding virtual device units (VDU). A scale in operation refers to cutting down the number of VDUs inside a VNF. The services provided by VNFM can be consumed by authenticated and properly authorized NFV management and orchestration functions. Thus, lots of research addresses the VNF management problem, and in the latter text, there will be an introduction for current solutions to the VNF LCM problems.

VNF scaling

VNF instance scaling is the result of a service quality threshold being crossed, in other words, whether because of the no longer acceptable service quality, requiring expanding capacity or because service quality and utilization are such that capacity can be contracted without affecting quality delivered. In general, the source which initiates the decision process is likely to come from:

- VNF, when it embeds a monitoring function or threshold crossing detection and event notification. The VNF instance might send the event to the EM, and decisions about actions may be implemented in EM and forwarded to VNF Manager. VNF may send the event directly to the VNFM as well.

- VNFM, when reception of a single VNF or infrastructure event might be sufficient to detect the need whether or not to scale, once this is done, the information on the event to monitor and the correlated scaling action will be provided in VNFD.

- VIM, when a monitoring function or threshold crossing detection is implemented in a VIM. The triggering events would be close to network congestion, the number of sessions, etc. VNFM will listen to the events and enforce the decision about actions.

- EM, when the monitoring function and threshold crossing detection and event notification is not in the VNF. In this case, decisions about actions might be implemented in EM and reported to VNFM.

- OSS/BSS, when the monitoring funct6ion and threshold crossing detection and event notification is not in the EM or crosses several EM. OSS/BSS would be both the detector and decision point

- OSS/BSS, when there is a change management process or capacity planning process based on traffic projections.

From the item sets above, we can group the scaling uses cases into three categories:

- Auto-scaling, in which the VNFM monitors the states of a VNF instance and triggers the scaling operation when certain conditions are met. In terms of monitoring a VNF instance's state, there are events that can be tracked in infrastructure-level and VNF-level events.

- On-demand scaling, in which a VNF instance or its EM monitor the state of the VNF instance and trigger a scaling operation through an explicit request to the VNFM.

- Management request scaling, where the scaling operation is triggered by some sender (OSS/BSS or operator) towards VNFM via the NFVO

The nature of the VNF and the measurement results that cross the thresholds will generally indicate the type of change required, which can be summarized as:

- configuration changes to the VDU instance
- add a new VDU instance (scale-out)
- shutdown and remove instances (scale-in)
- release resources from existing instances (scale-down)
- increase available network capacity
- provide increase bandwidth(or other network configurations)

Here we would like to use an example to illustrate the VNF scaling process. In the first example, we present an automatic VNF expansion flow case triggered by VNF performance measurement results. A VNF expansion refers to the addition of capacity which is deployed for a VNF. Expansion may result in a scale-out of a VNF by adding VNFCs to support more capacity or may result in a scale-up of virtualized resources in existing VNF/VNFCs. VNF expansion may be controlled by an automatic processor and may be manually triggered operation, as shown in Figure 2.5.

The main steps for the automatic VNF scale-up/scale-out are:

- The VNFM collects measurement results from the VNF (application-specific) using the operation notify or get performance measurement results of the VNF performance management interface.
- The VNFM detects a capacity shortage that requires the expansion (more resources).
- The VNFM requests granting to the NFVO for the VNF expansion based on the specifications listed in the VNFD (CPU, Memory, IP, etc.) using the operation Grant Lifecycle Operation of the VNF Lifecycle Operation Granting interface.
- The NFVO takes scaling decisions and checks resource requests (CPU, Memory, IP, etc.) against its capacity database for free resource availability.

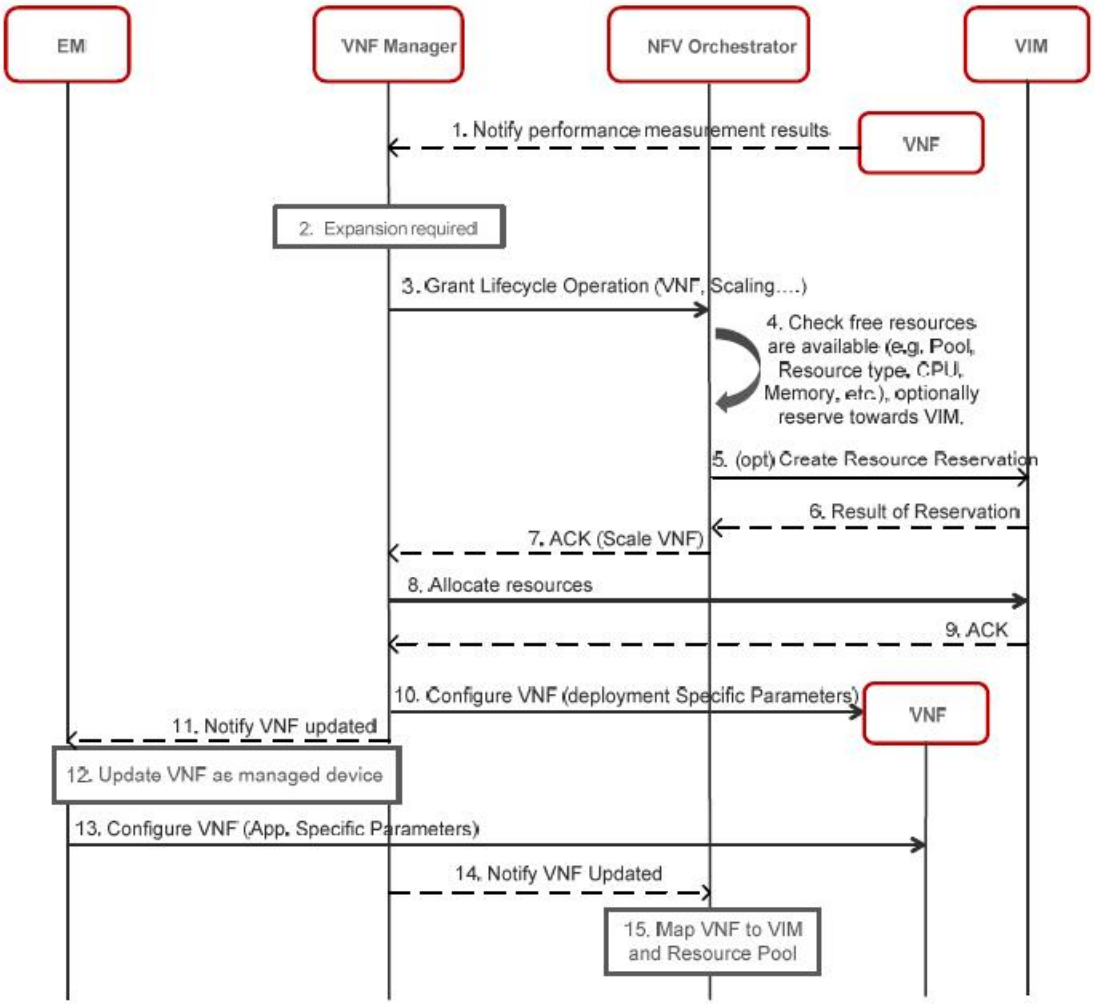

Figure 2.5: Automatic VNF scale-up/scale-out flow triggered by VNF performance measurement results[2]

- The NFVO may otherwise optionally make resource reservations for the requested resources by using the Create Resource Reservation operation over the Virtualised Resources Management interface.
- The NFVO grants the scale-out operation of the VNF to the VNFM and sends back sufficient information to further execute the scaling operation.
- The VNFM sends the request to create and start the VMs as appropriate. As instructed by the NFVO and inform VIM Identifier, VMs parameters are updated from those operations via Virtualised Resources Management interface.
- The VIM creates and starts the VMs and the relevant networking resources, then acknowledges successful operation to the VNFM.
- The VNFM configures VNF data specific for VNF instantiation using the add/create/set configuration object operations of the VNF Configuration interface.
- The VNFM notifies the EM that an existing VNF is updated with additional capacity using the VNF Lifecycle Change Notification interface.
- EM and VNF Manager update the VNF as a managed device.
- EM configures the VNF with application-specific parameters.
- VNF Manager reports successful VNF expansion to the NFVO using the VNF Lifecycle Change Notification interface. The NFVO is now aware that the new VNF configuration is instantiated in the infrastructure.
- The NFVO maps the VNF to the proper VIM and resource pool.

2.2.3 Virtualised infrastructure management

The VIM [34] is a functional block in the NFV-MANO framework, which lies at the bottom, responsible for managing and controlling the resources in an NFVI. The resources include the typical computing, storage and network capabilities provided in a cloud service provider throughout its NFV-Points-of-Presence (NFV-Pops). A VIM is likely to specialize in managing a certain type of NFVI resource(i.e., compute-concerned only) or multiple types of NFVI resources at once through a wide bound of interfaces interacting with. A VIM provides interfaces with network controllers to enable SDN controllers and hypervisors virtualization concerns as well as other northbound APIs for different use cases:

- Managing and orchestrating the allocation, upgrading, optimization, releasing, and reclamation of NFVI resources. Holding a global association with correlated compute, storage, and network resources in both edge-based physical servers and data-center-based ones. A VIM keeps an inventory to achieve this, with an illustration on a detailed illustration on the mapping between virtual resources and physical resources.

- Supporting the northern NFV-MANO functional blocks in creating, querying, updating, and deleting the VNFFGs; creating and maintaining the pre-defined VLs and network configuration in VNFDs and NSDs; managing the security group policies that provide network and traffic access control.

- Virtualised resources management. To manage and assign resources to the upper layer VNF and NS to full fill the capacity requirement.

2.3 Virtualization Technologies

In this section, we introduce two popular virtualization techniques: virtual machines and containers. Then we make comparisons and analyze their advantages.

2.3.1 Virtual machine

A virtual machine (VM) [35] is a virtual format or emulation of a physical computer. Virtualization makes it possible to create a virtual machine inside a physical machine, including a complete OS and related applications. VM cannot interact directly with the physical host, but via a lightweight software layer called a hypervisor to coordinate. The hypervisor helps allocate physical computing resources (i.e., processors, memories, and storage) to the VM. VM offers several benefits over traditional physical hardware:

- Resource utilization: since multiple VMs can run on a single physical server, users do not need a new physical computer when they need to operate on a different OS.
- Scale: with the idea of cloud computing, it is easy to deploy multiple copies of the same VM to better serve increases in load.
- Portability: VMs can be relocated as needed among physical servers in a network. This ensures the ability to allocate workloads to spare servers.

2.3.2 Container

Containers are lightweight executable units of software. The core technique is the kernel virtualization [3]. A container includes its basic application code as which is already packaged, as well as its libraries and dependencies. Multiple containers can share the same OS kernel instead of claiming a guest OS for each instance as in VMs. This greatly decreases startup time and the required resources for containers. Unlike VMs, containers do not need a dedicated hypervisor to virtualize physical hardware. Instead of the virtualization of the underlying hardware in physical servers, containers apply the virtualization in the

operating system itself (i.e., Linux kernel), making sure individual containers can include only the application and its corresponding dependencies and libraries. In IoT, containers are a more plausible option due to their lightweight, easier to deploy, and better resource utilization and version control than VMs. In NFV, containers also perform better because of their flexibility, especially in NS and VNF deployment. Figure 2.6 shows the difference between a host running containers and VMs [3].

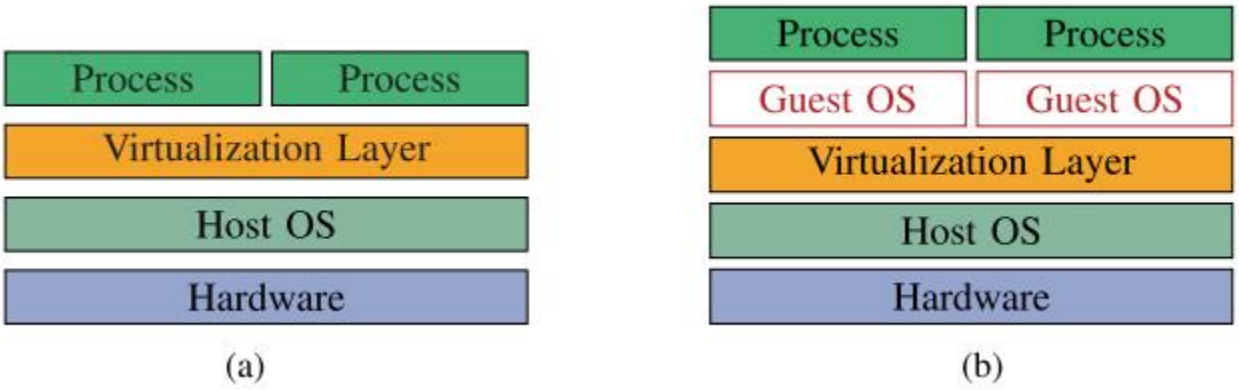

Figure 2.6: Comparison between container and VM [3]

2.4 VNF life-cycle management

In this section, we introduce VNF lifecycle management (VNF-LCM) related problems. Firstly we introduce the resource-concerned problems in VNF-LCM, then we introduce VNF placement problems in VNF-LCM. Then we compare the current VNF-LCM solutions.

2.4.1 Resource allocation in VNF-LCM

In VNF-LCM, resource allocation (RA) problem is one of the most common problems to be concerned. Generally, a physical network, SFC details and VNF specifications are given in a RA problem. There are three main step to formulate a RA problem:

- Calculate the number of VNFs that need to be optimized
- Place VNFs to the correlated NFV infrastructure under the requirement of SLA.
- Assign correlated service resource to the VNFs where the resources do not exceed the capacity of VNFs.

In most cases, the three steps are not independent of each other and those steps can be performed in different orders. For example, different scenarios of VNF placement may have an impact on the number of VNFs to be optimized. As a result, the solutions of the three steps are to be obtained simultaneously, which can make the RA problem more vivid and easier to understand.

In fact, the RA problem is NP-hard [36]. As a result, in a larger data center, the problem size is also much bigger (larger service chain, larger infrastructure). The cost of the RA problem can turn unaffordable. In order to solve this problem, several branches of solutions are developed including exact solutions to find global optimal target and approximation solutions to provide provable quality and run-time bounds. It is obvious to find that, exact solutions are targeting small-scale networks and can provide an optimal bound reference for heuristic solutions. Generally, RA problems can be formulated with optimization problems like Integer Linear Programming (ILP) or Mixed Integer Linear Programming (MILP), etc. ILP is typically NP-hard, while we can use approximation strategies to simplify the problem. For example, dynamic programming, branch and bound, cutting plane methods are common approaches. Approximation approaches make a trade-off between the exact optimal solution and the complexity of the algorithm, that is, it is possible to make the algorithm polynomial based on the strategy we select.

On the other hand, RA can be addressed with heuristic solutions. Heuristic algorithms can exploit problem-related knowledge where we cannot guarantee an optimal search for the problem [37]. A metaheuristic can formally define an iterative generation process that guides a subordinate heuristic by combining intelligently different content for exploring and exploring the research space, learning strategies are used to formulate information in order to find efficiently near-optimal solutions [38].

2.4.2 VNF placement problems

VNF placement problem is another hot topic in VNF-LCM. It is important to determine how to deploy the instance when infrastructure and physical resource is to be considered. Generally, service providers provide NSs and VNFs with the user demand, and VNFs run on its beneath infrastructure based on ETSI NFV architecture 2.1. Service providers should be responsible for each physical node and the VNF instances which are running on the physical server. Therefore, the service provider is responsible for selecting a suitable physical node to host the VNF. Since most VNFs are represented by VMs, the VNF placement problem can also be represented by the VM placement problem. There are lot of literature addressing the VM placement problem. For example, in [39], the author presents a traffic-aware VM placement problem to improve the scalability in data center network and proposes a two-tier approximation algorithm to improve the efficiency and overcome the large problem size.

2.4.3 Current VNF LCM problem solutions

A large number of existing research efforts have addressed different functionalities related to the LCM of VNFs, such as VNF placement and migration, scaling, and fault management. We first survey a sample of these approaches and then discuss related efforts that employ ML for VNF management. On the other hand, there are lots of heuristic solutions. Bari et al. [40] propose a DP-based heuristic to solve the RA problem with a large number of instances. Metaheuristic solutions such as simulated annealing, tabu search, genetic algorithms, etc. can be performed on RA problems to find better solutions. In [41] the author uses a genetic algorithm to simulate the dynamic Deep Packet Inspection (DPI) deployment problem, which provides a trade-off between the number of engines and the network load to minimize the global cost of the deployment. The resource allocation problem is always addressed in academia. Taleb et al. [42] use ILP to formulate VNF placement problem in a mobile network. In [36], the author presents an ILP formulation and uses an approximation algorithm with proven performance to reduce the problem to Generalized Assignment Problem (GAP).

The problem of VNF resource reservation has been addressed extensively in the literature [20]. For example, Roy et al. [43] proposed a predictive approach to minimize the resource provisioning costs while guaranteeing the application QoS [44]. Similarly, Rao et al. [45] proposed a resource allocation scheme that guarantees QoS applications. Joint VNF placement and resource allocation has been considered by Zhang et al. [46]. The authors proposed a novel VNF chain placement solution that maximizes resource utilization and minimizes request-response latency. Similarly, Ghaznavi et al. [47] proposed an optimal placement solution of VNF instances according to demanded workload. The main limitation of VNF placement and resource allocation schemes, in general, is that they assume that VNF resource demands can be modeled as a simple scalar value, which does not account for resource demand fluctuation. This complicates the management operations by the VNFM.

A large body of research efforts has also focused on vertical and horizontal scaling. For example, Li et al. [48] analyzed the advantages and disadvantages of vertical scaling and demonstrated that some VNFs could not improve their performance through vertical scaling. On the other hand, Rankothge et al. [49] illustrated that startup time is costly when horizontally scaling new VNFs. Yu et al. [50] proposed a fine-grained hybrid scaling scheme for SFCs to achieve NFV scaling efficiency while minimizing resource cost for each SFC. Similarly, in ElasticNFV [51], the resource consumption of the hosting VM is monitored in order to achieve a balance between vertical scaling and migration whenever there are conflicts in resource demands. Our proposed work is complementary to the approaches mentioned above; once optimal action and action parameter pairs have been selected, any existing scaling or migration schemes can be used to actuate the desired action.

Finally, it is worth noting that the adoption of ML schemes for various LCM functionalities has been extensively studied in the Literature. For example, z-TORCH [52] exploits machine-learning-based techniques in NFV entities orchestration. They provide a limited number of VNF KPIs and apply Q-learning in their monitoring mechanism. Similarly, the work in [21] studies the VNF scaling and migration problems to meet the delay require-

ments in the presence of nonstationary traffic. Machine learning schemes have also been used recently for VNF resource prediction [53, 54] and for optimal paths for SFCs in [55]. The work in [56] presents a PAMDP on solving 5 G-related problems using reinforcement learning. Finally, Lange et al. [57] provide a comprehensive analysis of existing ML-related VNF management approaches and then propose a general network intelligence architecture for VNF that focuses on using ML for several separate functionalities such as anomaly detection, VNF placement, and chaining.

The aforementioned approaches have demonstrated the applicability of ML schemes as powerful tools for various VNFM functionalities. However, our proposed work provides a holistic ML-based solution that can bring full automation of the VNFM functionalities.

2.5 Reinforcement learning

Reinforcement learning (RL) [58] is a technique that refers to both a learning problem and a branch of machine learning [59]. A learning problem is a set of issues that an agent tries to learn to control a system known as an environment to maximize some numerical value, or in other words, a long-term objective, known as discounted cumulative reward signal. In this section, we introduce the methods and tools needed to analyze stochastic dynamical systems. Then we review the major solutions to reinforcement learning algorithms and major extensions on RL algorithms with the deep learning method.

2.5.1 Analysis of dynamic systems

RL performs well in optimizing controlling dynamical systems [60]. For example, a controller, which is known as an agent, receives the updated state in the system, and feedback, which is called a reward, is associated with the state received. The next step for the agent is to execute a control signal or to take action to send it back to the system. In response to the agent, the system will make a new state transition, and the process will be repeated until the agent successfully makes it reach an optimal state or be asked to stop. The goal is to seek a proper method to control the system to maximize the total reward in episodes.

An agent is a term referring to the controller, and the environment is the corresponding

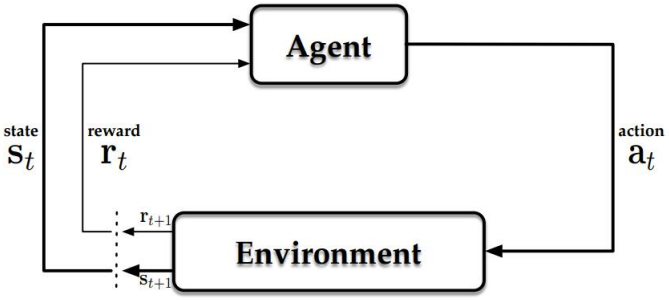

Figure 2.7: High-level stochastic dynamical system schematic(Sutton and Barto, 1998).[4]

system for the agent. The goal of an RL agent is to be trained in the environment and successfully interact with the environment so as to maximize the scalar objective over time. An action $a_t \in \mathcal{A}$ refers to the control signal when the agent pushes it back to the system, as it is the only method for the agent to make an influence on the environment state and in return, leading to another state to be observed and a reward to be obtained at time slot t. The action space $\mathcal{A}$ is a set of actions that the agent is able to take and can be represented as discrete and continuous:

- discrete: $\mathcal{A} = a_1, a_2, ..., a_M$;
- continuous: $\mathcal{A} \in [c, d]^M$, where c and d is the lower bound and upper bound of a continuous action value

A reward $r_t \in \mathcal{R}$ is the feedback from the environment, indicating the performance of the agent at the time slot for its previous action selected. The goal of the RL agent is to maximize the cumulative reward over a set of training episodes. RL focuses on sequential decision-making tasks. In terms of different training scenarios, the RL agent either optimizes delayed rewards for long-term goals or focus on the immediate reward instead. This property of RL is important for problems in some areas, such as financial applications, since investment horizons will range from days to weeks to a long term. If the term is long, the myopic agents will be likely to perform poorly because the evaluation of long-term rewards matters in order to obtain higher gains [61]. On the other hand,

the applicability of RL depends on the hypothesis, that is, to describe all goals by the maximum expected cumulative reward. As a result, the selection of the proper reward function for each action and state pair is an essential task for RL. It is a reflection point for the RL agent to determine whether the action is acceptable or not.

State $s_t \in \mathcal{S}$ is the fundamental component of RL. In general, the state can be represented by the agent state and the environment state. For an RL agent, because it cannot directly interact with the state, a support observation, $o_t \in \mathcal{O}$, is introduced to represent the current state after a specific action. The environment state s_t^e represents the internal system, to determine the next observation o_{t+1} and reward r_{t+1}. This state is not visible to the agent in most cases and contains irrelevant information. In contrast, agent state s_t^a is the representation from the agent to the environment. Here we would like to introduce history $\vec{h_t}$ at time slot t, which is the trace of the sequence of actions, observations, and rewards over time, and can be shown in the equation:

$$\vec{h_t} = (a_1, o_1, r_1, a_2, o_2, r_2, ..., a_t, o_t, r_t) \tag{2.1}$$

The agent state can be any function f in the history:

$$s_t^a = f(\vec{h_t}) \tag{2.2}$$

With all possible states, state space $\mathcal{S}$ can represent the entire possible combinations of states could appear that can be either discrete or continuous similar to the action space:

- Discrete states space: $\mathcal{S} = s_1, s_2, s_3, ..., s_n$
- Continuous states space: $\mathcal{S} \subset \mathcal{R}^{\mathcal{N}}$

To better observe states changes, a fully observable environment is needed:

$$o_t = s_t^e = s_t^a \tag{2.3}$$

In our case, f is the function that is unknown to the agent but has access to the observation o_t while not the environment state s_t. However, it is not always possible to observe all the

environment parameters at the same time. As a result, partially observable environments provide indirect accesses to the environment state. Thus, the agent will have to build its state by using the following methods:

- History from observations: $s_t^a \equiv \vec{h_t}$;
- Recurrent neural network: $s_t^a \equiv f(s_t^a, o_t, \theta)$

Through updating the basic dynamic system in Figure 2.7, we need to take partial observability into account. We use R_s^a and $P_{ss'}^a$ to represent the reward generation function and the probability transition matrix function of the Markov Decision Process (MDP), respectively. We can treat the system as a probabilistic graphical model so that state s_t will be latent variable to be either deterministic or stochastic, which depends on the nature of f, and the previous factors determine the observation o_t. In a partially observable environment, the agent has to re-construct the environment state by either using the complete history h_t or a stateful sequential model.

2.5.2 Component of reinforcement learning

An RL agent contains one or more of the following components:

- Policy: actions agent takes at a certain state, can be derived as a function
- Value Function: the performance evaluation for one state or a state-action pair
- Model: the representation of the environment from the agent.

These components have great impact on the RL agent design as well as the RL algorithm. First of all, a concept return G_t should be introduced, which is the future discounted reward at time slot t:

$$G_t = r_0 + \gamma r_1 + \gamma^2 r_2 + ...\gamma^t r_t \tag{2.4}$$

On the other hand, a policy π refers to the behavior of an agent, more precisely, it is a reflection from a certain state to the next action the agents takes, such that:

$$\pi : \mathcal{S} \to \mathbf{A} \tag{2.5}$$

where $\mathcal{S}$ is the state space and $\mathbf{A}$ is the action space. By far, we can present a policy function as such:

- Deterministic policy: $A_{t+1} = \pi(s_t)$
- Stochastic policy: $\pi(a|s) = \mathbf{P}[a_t = a|s_t = s]$

Then we introduce state-value function v_π, which is the expectation of the return G_t when taking a policy π and transit from state s:

$$v_\pi : \mathcal{S} \to \mathbf{R}, v_\pi(s) = \mathbf{E}[G_t|s_t = s] \tag{2.6}$$

where $\mathcal{S}$ is the state space and $\mathbf{R}$ is the reward function. Also we have action-value function, q_π, which is the expectation of the return G_t, starting from state s, upon taking action a, then followed by a policy π, such that:

$$q_\pi : \mathcal{S} \times \mathbf{A} \to \mathbf{B}, q_\pi(s, a) = \mathbf{E}[G_t|s_t = s, a_t = a] \tag{2.7}$$

Markov Decision Process

We then introduce Markov Decision Process (MDP) [62]. An MDP refers to a specific type of discrete-time stochastic dynamical system. An MDP has such property that it can converge to an optimum global policy. To some extent, it can be transferred to describe any dynamical system to provide a robust and assertive representation framework for most stochastic processes as well as a standard way of adjusting dynamical systems. In the latter text, we will introduce the Markov Property and a detailed definition of an MDP, as well as its optimality property.

In general, a state s_t satisfies the Markov property iff:

$$\mathbf{P}[s_{t+1}|s_t, s_{t-1}, ..., s_1] = \mathbf{P}[s_{t+1}|s_t] \tag{2.8}$$

where it implies the previous state s_t is sufficient enough for future statistics and the future state is only determined by the current state. As a result, the longer-term history $\vec{h_t}$ can be discarded. All the fully observable environment has the property, in other words, if it satisfies Equation 2.8, then it can be stated as a MDP. An MDP has the following component:

- State $\mathcal{S}$, states space that satisfy the Markov property;
- Action $\mathbf{A}$, action space;
- Probability Matrix $\mathcal{P}$, state transition probability matrix;
- Reward $\mathcal{R}$, reward function in each episode or step;
- γ, is a discount factor;

An MDP can be optimally solved using the value function and best policy the agent has found. The optimal state-value function V_* is the maximum of all policies as well as optimal action-value function Q_*:

$$\mathcal{V}_*(s) = \max_\pi \mathcal{V}_\pi(s), \forall s \in \mathbf{S} \tag{2.9}$$

$$\mathcal{Q}_*(s, a) = \max_\pi \mathcal{Q}_\pi(s), \forall s \in \mathbf{S}, a \in \mathbf{A} \tag{2.10}$$

Bellman Equation

If an MDP is given, due to the Markov property stated in 2.8, the policy π can be represented as a distribution over actions given states:

$$\pi(s|a) = \mathbf{P}[a_t = a|s_t = s] \tag{2.11}$$

We assume that the policy π is stochastic due to the state transition probability matrix $\mathcal{P}$ [63]. Because of the Markov property, MDP policies only depend on the current state and they are time-independent and stationary:

$$a_t \sim \pi(\cdot|s_t), \forall t > 0 \tag{2.12}$$

On the other hand, the state-value function v_π can also be parted into the discounted reward of the next state γr_{t+1} and the immediate reward:

$$\begin{aligned} \mathcal{V}_\pi(s) &= \mathbf{E}_\pi[G_t|s_t = s] \\ &= \mathbf{E}_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + |s_t = s] \\ &= \mathbf{E}_\pi[r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3}|s_t = s] \\ &= \mathbf{E}_\pi[r_{t+1} + \gamma v_\pi(s_{t+1})|s_t = s] \end{aligned} \tag{2.13}$$

The same applies for action-value function $\mathcal{Q}_\pi$:

$$\mathcal{Q}_\pi(s, a) = \mathbf{E}_\pi[r_{t+1} + \gamma q_\pi(s_{t+1})|s_t = s, a_t = a] \tag{2.14}$$

The equations above are the Bellman Expectation Equations for Markov Decision Processes.

Value-based reinforcement learning

We introduce the basic value-based reinforcement learning approach[64] in this subsection. To address the problem, in most cases, the environment information of the MDP is essential to the RL agent. However, there are problems that the transition probability is difficult to obtain. Thus, model-free RL models are developed, Q-learning, as an instance. Q-learning is introduced as a classic RL algorithm to solve related problems. Q-value is a state-action function to represent the current behavior of the agent. In general, Q-value is stored in a Q-table, which contains the full map of state space and action space, and thereby, Q-learning has more use cases in a simpler problem. An action-value function $Q_\pi(s, a)$ can

be represent as:

$$Q_\pi(s,a) = \mathbf{E}_\pi[r_{t+1} + \gamma q_\pi(s_{t+1}|s_t = s, a_t = a] \tag{2.15}$$

For an optimal action value function $Q^*(s,a)$ for all state-actions pairs, the optimal policy π^* can be represented by:

$$\pi^*(s,a) = \begin{cases} 1 & a = \arg\max_a[Q(s,a)] \\ 0 & otherwise \end{cases} \tag{2.16}$$

Then we can update the problem to a optimal Q-value seeking problem. Which can be represented as:

$$Q_{t+1} = Q_t(s,a) + \alpha_t[r_t(s,a) + \gamma \max_a Q_t(s',a') - Q_t(s,a)] \tag{2.17}$$

Where α is the learning rate value and γ is the decaying factor. To update the Q-value, the essence is to calculate the temporal difference (TD) between the predicted Q-value and the current value. For deep Q network(DQN) [65], which has better performance than the plain Q-learning in terms of the size of actions and states. Instead of a Q-table, a neural network is applied to approximate the value of $Q^*(S,A)$. When a non-linear function approximator is applied in the model, the average reward might not be converged, which indicates the massive influence on the Q-value selection, resulting in different policies. Thus, distribution and correlations of data between the current Q-value and the target one are not converged. We can use experience replay and target Q network to solve the problem. Experience reply is a mechanism that initializes a replay memory $\mathbf{D}$ which is called a replay buffer, storing transitions (s_t, a_t, r_t, s_{t+1}) and experiences which are generated with the ϵ-greedy strategy. Unlike the Q-learning feeding successive transitions into a mini-batch, DQN will select transitions from the replay buffer consisting of a mini-batch to train the deep neural network. And the deep neural network will update the Q-values to new transitions. When the memory buffer is fully updated, the upcoming transitions will replace the old experiences directly, ensuring the training process at an efficient pace in DNN by applying

both old and new experiences. Meanwhile, the replay memory will guarantee a more independent and identical transition to solve the distributed and the correlation problems between observations. On the other hand, DQN also introduces a fixed target Q-network in the algorithm. During the training process, the action-value function will be shifted. As a result, it will be likely that the action-value function gets out of control if constantly shifting is applied to update the Q network. In consequence, the algorithm will result in destabilization. The target Q network will be updated infrequently to stay stable in periods to converge the target Q network to address the problem. However, DQN is still limited in some cases. One core problem is the slow training process and massive complexity. Another problem is the over-estimations within the training process, leading to bad performance. Because of the maximum action value selection as the approximation of the maximum expected action value in Q-learning, overestimations are produced with the introduction of a positive bias. Specifically, when Q values are not converged and inaccurate in the initial learning process, if the action is applied with an over-estimated model, it is likely to influence the latter learning process. Thus, to address the problem, Hassel et al. [66] proposed a method by applying two Q learning algorithms Q_1 and Q_2, to select action values simultaneously through a lost function:

$$r_j + \gamma \mathcal{Q}_2\left(s_{j+1}, \arg\max_{a_{j+1}} \mathcal{Q}_1(s_{j+1}, a_{j+1}; \theta_1); \theta_2\right) - \mathcal{Q}_1(s_j, a_j; \theta_1) \tag{2.18}$$

It is notable that the online weight θ_1 is the core factor for action selection. As an indication, a greedy strategy is always used to update the current values, as defined by θ_1. However, θ_2, which is the weight parameter, is used to give the Q-value of the policy. The weight parameter can be updated symmetrically by switching between θ_1 and θ_2. With this idea, double DQN is proposed with the loss function update rules:

$$r_j + \gamma \hat{\mathcal{Q}}\left(s_{j+1}, \arg\max_{a_{j+1}} \mathcal{Q}(s_{j+1}, a_{j+1}; \theta); \theta'\right) - \mathcal{Q}(s_j, a_j; \theta) \tag{2.19}$$

where we replace the weight network θ_2 with the target network θ'

Policy-based reinforcement learning

We introduce the basic policy-based reinforcement learning approach [67]. To be distinguished from the value-based one, we use policy π_θ to represent the correlations between states and actions. Generally, policy gradient (PG) is widely used in policy-based RL. The policy parameter θ, refers to a function estimator that derives the possible parameter that could be added with the action itself. Policy search methods directly learn to estimate the policy pi_θ. In order to maximize an objective function, neural networks are often used in policy-based RL, and we can use the following equation to represent the estimator function:

$$J(\theta) = \mathbb{E}_{\tau\sim\rho_\theta}[R(\tau)] = \mathbb{E}_{\tau\sim\rho_\theta}\left[\sum_{t=0}^{T}\gamma^t r\left(s_t, a_t, s_{t+1}\right)\right] \tag{2.20}$$

where $\mathbb{E}_{\tau\sim\rho_\theta}$ represents the expectation of return by iterating all possible trajectories. The likelihood function can be represented using the trajectories:

$$\rho_\theta(\tau) = p_\theta\left(s_0, a_0, \ldots, s_T, a_T\right) = p_0\left(s_0\right)\prod_{t=0}^{T}\pi_\theta\left(s_t, a_t\right)p\left(s_{t+1} \mid s_t, a_t\right) \tag{2.21}$$

where $p_0(s_0)$ represents the probability of the initial state $s0$ and $p(s_{t+1}|s_t, a_t)$ denotes the probability of the state-action transition. The objective function in Equation 2.20 can be rewritten by integrating the distribution of the trajectories:

$$J(\theta) = \int_\tau \rho_\theta(\tau)R(\tau)d\tau \tag{2.22}$$

Within the process, we use Monte-Carlo (MC) sampling to present a the estimation of the objective function. By sampling the trajectories periodically, we can calculate an averaged returned:

$$J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} R\left(\tau_i\right) \tag{2.23}$$

Then we apply the gradient ascent method on the weights θ to maximize the objective function $J(\theta)$. With this idea, we can focus on the gradient value $\nabla_\theta J(\theta) = \frac{\partial J(\theta)}{\partial\theta}$. Once an estimation of the policy gradient is obtained, gradient ascent method will be updated

directly $\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$, where η is the learning rate.

In [68], the author presents an effective estimation of policy gradient. By considering the obtained return $R(\tau)$, we can calculate the policy gradient independently:

$$\begin{aligned} \nabla_\theta J(\theta) =& \nabla_\theta \int_\tau \rho_\theta(\tau) R(\tau) d\tau \\ =& \int_\tau \left(\nabla_\theta \rho_\theta(\tau)\right) R(\tau) d\tau \\ =& \int_\tau \rho_\theta(\tau) \nabla_\theta \log \rho_\theta(\tau) R(\tau) d\tau \end{aligned} \tag{2.24}$$

Where log-trick is applied in the equation $\rho_\theta(\tau)\nabla_\theta \log \rho_\theta(\tau) = \rho_\theta(\tau)\frac{\nabla_\theta \rho_\theta(\tau)}{\rho_\theta(\tau)} = \nabla_\theta \rho_\theta(\tau)$. As a result, we can rewrite the expectation in the following text:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \rho_\theta} \left[\nabla_\theta \log \rho_\theta(\tau) R(\tau)\right] \tag{2.25}$$

Similarly, with Equation 2.21 we can also rewrite the log version of likelihood function of a trajectory:

$$\log \rho_\theta(\tau) = \log p_0\left(s_0\right) + \sum_{t=0}^{T} \log \pi_\theta\left(s_t, a_t\right) + \sum_{t=0}^{T} \log p\left(s_{t+1} \mid s_t, a_t\right) \tag{2.26}$$

Since $\log p_0$ and $\log p\left(s_{t+1} \mid s_t, a_t\right)$ is irrelevant with the parameters θ, the gradient of the log version of likelihood function can be simplified as:

$$\nabla_\theta \log \rho_\theta(\tau) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta\left(s_t, a_t\right) \tag{2.27}$$

Now we can represent the policy gradient by:

$$\begin{aligned} \nabla_\theta J(\theta) =& \mathbb{E}_{\tau \sim \rho_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta\left(s_t, a_t\right) R(\tau)\right] \\ =& \mathbb{E}_{\tau \sim \rho_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta\left(s_t, a_t\right) \left(\sum_{t=0}^{T} \gamma^t r_{t+1}\right)\right] \end{aligned} \tag{2.28}$$

By applying Monte-Carlo (MC) sampling, the policy gradient can be estimated straight-

forward.

Here, we will introduce the policy gradient theorem [69]. As discussed in early context, the basic logic behind the policy-based RL is to modify the parameter θ of the policy in the direction of the performance gradient $\nabla_\theta J(\pi_\theta)$. In [70], the author proves the idea that the policy gradient can be estimated by substituting the return of the sampled trajectory using the state-action pair from the Q function. We can present the policy gradient theorem is the following text:

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \int_{\mathcal{S}} \rho^\pi(s) \left(\int_{\mathcal{A}} \nabla_\theta \pi_\theta(a|s) \mathcal{Q}^\pi(s,a) da \right) ds \\ &= \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta} \Big[\nabla_\theta \log \pi_\theta(a|s) \mathcal{Q}^\pi(s,a) \Big]\end{aligned} \tag{2.29}$$

where ρ^π is the distribution of reachable states under policy π. Since the actual return $R(\tau)$ is substituted with $\mathcal{Q}^\pi(s,a)$, we can calculate the policy gradient by observing the expectations over single transitions, which allows bootstrapping as in TD methods.

The straightforward policy gradient method has a set of problems in many areas. The primary concern is the high variance of the policy gradient, and a typical example is the REINFORCE algorithm [71]. To address the problem, the policy gradient theorem provides an actor-critic architecture to learn parameterized policies. Detailed speaking, if the action-value function can be estimated accurately and used to update the policy, there will be profound discoveries in the actions space. An actor-critic architecture is composed of an actor-network and critic network. An actor is a network that gives the appropriate action based on the current states, and a critic is a network that estimates the Q-value function. In deep reinforcement learning, non-linear neural network function approximators can represent actor-network and critic network. Moreover, the actor uses gradients derived from the policy gradient method to update the policy parameters. In order to reduce the variance for further update, a another feasible approach is to subtracting a baseline function from the Q function, which in general, can be represented by the value function $\mathcal{V}^\pi(s)$. Thus, we define an advantage function A with indicators judging whether better

performance is made:

$$A^{\pi}(s,a) = \mathcal{Q}^{\pi}(s,a) - \mathcal{V}^{\pi}(s) \tag{2.30}$$

We can simplify the problem by calculating the two estimation functions: $\mathcal{Q}^{\pi}(s,a)$ and $\mathcal{V}^{\pi}(s)$. Then we can rewrite the policy gradient problem:

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{s\sim\rho^{\pi}, a\sim\pi_{\theta}}\Big[\nabla_{\theta} \log \pi_{\theta}(a|s) A^{\pi}(s,a)\Big] \tag{2.31}$$

There are different methods that can be used to estimate the advantage function $A(s,a)$:

- $A(s,a) = r(s,a) + \gamma\mathcal{V}(s') - \mathcal{V}(s)$ is the **TD advantage estimate or TD error**.
- $A(s,a) = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n \mathcal{V}(s_{t+n+1}) - \mathcal{V}(s_t)$ is the **n-step advantage estimate**.
- $A(s,a) = G(s,a) - \mathcal{V}(s)$ is the **MC advantage estimate**, the action-value function is replaced by the actual return.

Deep Deterministic Policy Gradient [72] is another successful algorithm that utilizes the performance of policy gradient by a model-free off-policy actor-critic architecture. Even though DQN can solve problems with high-dimensional state spaces, the problem is the discrete state space and low-dimensional action space. Lillicrap et al. [73] derived a special actor-critic algorithm with a model-free mechanism. The algorithm has great performance in continuous action spaces. With the idea of the deterministic policy gradient (DPG) theory [74], the DPG algorithm outputs a deterministic policy $\mu_{\theta}(s)$ with parameter θ, were presenting a reflection between a specific state and action. If the state space is continuous, it can be naturally derived from an indication that the gradient of the objective function is the same as the gradient of the Q-value. We can rewrite an unbiased estimate $\mathcal{Q}^{\mu}(s,a)$ to update the policy $\mu_{\theta}(s)$ to the direction of $\nabla_{\theta}\mathcal{Q}^{\mu}(s,a)$, leading to a return with an associated action:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{s\sim\rho_{\mu}}\Big[\nabla_{\theta}\mathcal{Q}^{\mu}(s,a)|_{a=\mu_{\theta}(s)}\Big] \tag{2.32}$$

where ρ_{μ} represents the distribution of reachable states by the policy. Then we can extend

the chain rule to the gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho_\mu}\left[\nabla_\theta \mu_\theta(s) \times \nabla_a \mathcal{Q}^\mu(s,a)|_{a=\mu_\theta(s)}\right] \tag{2.33}$$

According to the update rule, a deep deterministic policy gradient(DDPG) is developed to extend the DPG theory with non-linear function approximators. By combining the idea of DQN and DPG, the critic network uses an experience replay memory buffer and target network for action selections. If applied with a slow tracking on the learned networks, better performance could be found:

$$\theta \leftarrow \tau\theta + (1-\tau)\theta' \tag{2.34}$$

This indicates that the target values are always late concerning the trained networks. This would ensure the stability of the learning process. It is notable that the biggest challenge of learning in a continuous action space is exploration. The exploration policy μ' can be constructed by adding white noises to the actor policy to address the problem.

2.5.3 Examples of RL approaches

In [75], the author proposed a model that is able to investigate a multi-user and one edge server MEC scenario. A deep Q network is proposed in the paper to optimize the offloading decisions and resource allocation problem jointly. Another example [76] addresses the problem of multiple mobile edge cloud servers scenarios. The author proposed a DRL-based scheme for an IoT device with energy harvesting (EH) considering. The offloading computation process can be regarded as an MDP where the IoT devices select the offloading location under the condition of current system states(battery status, previous ratio transmission rate) and then predict the amount of harvested energy. Chen et al. [77] proposed a double DQN method to address a MEC scenario where mobile users in an ultra-dense sliced radio access network (RAN) with multiple base stations. The author provides an online strategy to compute offload consumption with a Q-function decomposition technique to maximize long-term performance.

2.6 Summary

This chapter discusses the fundamental network function virtualization component and briefly introduced the management and orchestration panel in NFV. We stressed introducing the VNFM and discussed its functionalities and related problems. We present the virtualization techniques to make comparisons on VM and containers. And then, we introduced the basic definition of RL and discussed the related component in RL, i.e., Markov decision process and Bellman Equation. In the latter context, we summarized two basic formats of RL problems:value-based and policy-based. For value-based RL, the most classic algorithm is the Q-learning algorithm. We introduced it and made an extension to a higher level with other Q-learning-based algorithms. On the other hand, we also introduced policy-based RL. Policy gradient is the classic one as a summary algorithm. Besides, we introduced the essential component of deep reinforcement learning extensions and summarized some typical problems like reward shifting and policy distillation. Finally, based on the techniques we addressed, we design a novel model for VNF management and orchestration by using reinforcement learning algorithms to automatically and efficiently deal with life-cycle management problems of VNFs in the following chapter.

Chapter 3

Proposed VNF Manager Framework

In this chapter, we propose our system model and present our algorithms. section 3.1 presents the system architecture. Section 3.2 formulates the VNF structure. We then introduce the detailed design of our proposed RL problem model in Section 3.3, and present our algorithms. In Section 3.5 we present the action space of our model. The core problem is to determine a novel method to address all possible state transitions and find a solution to optimally assign resource to the VNF. In Section 3.6 We introduce a parameterized reinforcement learning algorithm to find the solution with both value-based and policy-based reinforcement learning methods which are executed on an edge Kubernetes server. We summarize the chapter in Section 3.7.

3.1 System architecture

In this section, we describe our VNFM within the context of the ETSI NFV [12] as shown in Figure 3.1. The Management and Orchestration (MANO) module represents the main component within the ETSI NFV architecture [12] as shown in Figure 2.1. The NFV infrastructure contains the physical resources at the lower layer of the architecture and represents the available compute, storage, and network resources. These physical resources host the VNFs, which represent the second layer, using different resource visualization approaches. These two layers are managed by two corresponding modules in MANO.

The NFVI layer is managed by the containerized infrastructure manager (cIM), while the VNFs' resource consumption monitoring, life cycle, performance, and fault management is performed by the VNFM. The VNF monitoring module in the VNFM is responsible for monitoring the resource consumption dynamically and analyzing collected data. On the other hand, the NFVO maintains catalogs describing available NSs as well as NVF descriptors. The latter describes the decomposition of each VNF and the possible flavors or configurations for each component.

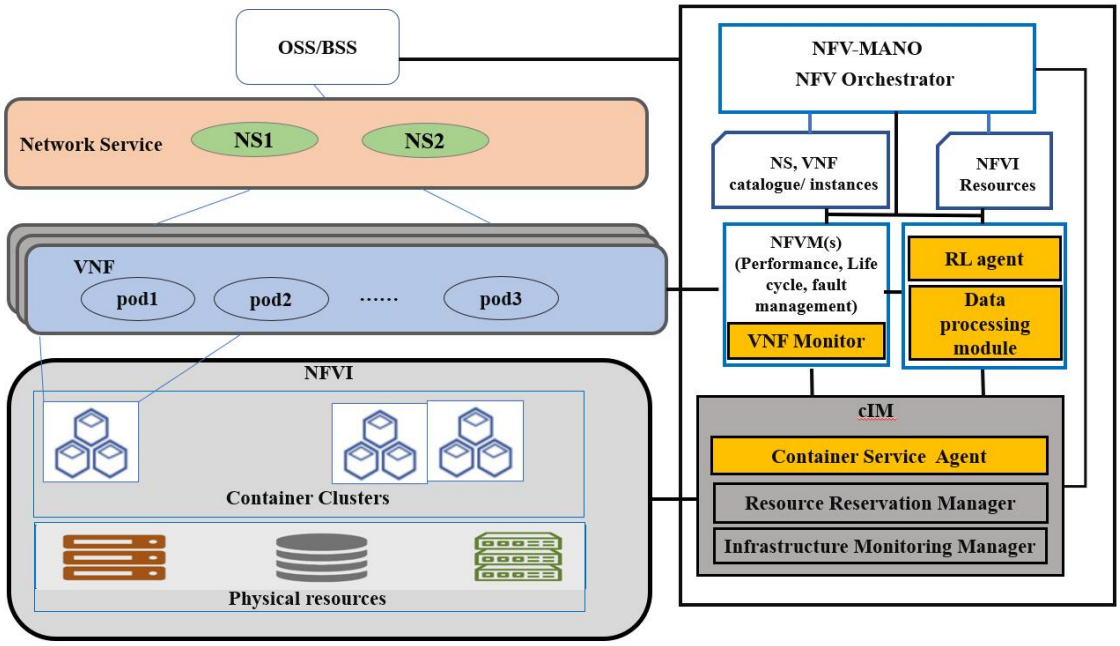

Figure 3.1: NFV framework with RL agent

We focus on the management and orchestration of a single VNF comprised of multiple VNF components referred to as the virtual deployment units (VDUs). Monitored measurements that are collected from the VNFI or by the VNFM are consolidated together to form an observed state of the VNF. A VNF modeler compares the observed state to the VNF desired state that is described by the SLA [18] and reports to the RL agent. The latter decides on an appropriate LCM parameterized action and may communicate with the VNFO or the cIM.

In the next section, we focus on the functionalities of the VNF modeler and the PASM action selection process. However, it is worth noting that once an action is selected,

our proposed VNFM can employ existing or novel VNF management algorithms (e.g., algorithms for migration, scaling, and fault management) to execute the selected action.

3.2 Structure of a VNF

3.2.1 Virtual device units and VNF component

As discussed in the introduction chapter, a VNF represents the virtualization of a network function, where the function behavior and states are independent of whether the network function is virtualized or not, while the external behavior of the component should be identical in either case.

Each VNF is composed of one or multiple components called the VNF component (VNFC). A VNFC is mapped with virtual machines and containers. A VNF can be deployed on top of single virtual machines or multiple virtualized instances instead to host every single component of the VNF in complete isolation for elastic management and orchestration operations.

In the VNF descriptor file, a configuration contains the inner component of the VNF, including one or more virtual deployment units (VDUs) and entities representing a single unit of deployable VNFCs. A VNFC refers to a composition comprising the VNF, and a VNFD describes how the VNFC instances are bind together to create a complete VNF and their correlations between VNFCs through connection points (CPs). A single component generally matches with a single VM under the control of a VIM instance.

A VDU contains basic information, for example, the base image of the instance to be used when instantiating a VNFC instance, the configuration parameter, and mandatory deployment requirement to create the components, and the amount of the resource units to be deployed and available resource to be scaled.

3.2.2 Virtual links

Virtual links are another essential requirement to define a VNF. It is considered as a configuration component reflecting how the deployed VNFCs connect with each other after been deployed into an NFVI.

Virtual links are specified in the VNFD configuration file, all together with CPs of the deployed instances. These entities represent the interconnections of the VNFCs with each other and with the outside network, shown in Figure 2.7. The connectivity options exposed to the NFVO are described in the virtual link descriptors (VLDs). Meanwhile, the NFVO also obtains information from a VNF forward graph, which can be considered as a graph containing the interconnections among all the VNFs. In the graph architecture, the data coming from the instances will be delivered through to a lower-level system to enable logic configuration of pre-existing hardware and software network components, which has a similar existing format of SFC.

The implementation of the above components in the infrastructure network is dependent on the physical locations of the end server and the implementation nature of the technology itself. If two VNFs share one hypervisor on the server, these two VNFs could be connected using a virtual switch under the virtual technology or otherwise are based on an external Ethernet switch. The VLD contains a description of each virtual link to determine where the VNF should be placed according to the current NFVI. These can be considered as a mapping configuration between the VIM layer and the VNFM layer. The VIM will use the VLD information to establish the proper paths and virtual local area networks (VLANs) in terms of use cases by using the basic topology described in the VLDs.

As an example, the VL include constraints and requirements on:

- The bandwidth of the link (the capacity of the link can offer)
- The QoS expected from the link, the value of jitter and latency requirement from the channel
- passive monitoring information, active loopbacks at the endpoints configuration

Similarly, a link can be internal defined, which is corresponded to the internal communications between VNFCs.

3.3 Proposed VNF Model

In this section, we introduce how we formulate our VNF model. Consider a NFVI with limited resources $\mathcal{R}_{pool}$ under a permanent constraint:

$$r \leq \mathcal{R}_{pool} \tag{3.1}$$

Where r is the current resource vector (cpu,mem,$throughput$,$latency$) of a VDU and each item in the resource vector cannot exceed the maximum value in $\mathcal{R}_{pool}$. An operator offers NSs as chains of VNFs. Each VNF (e.g., a virtual router or a customer premises equipment) is offered as part of an NS catalog and is described as a set of connected VDUs (e.g., a software meter, a policier, a packet marker and a router). In turn, each VDU type s in the NS catalog S is associated with a set:

$$f_s = \{f_{s1} \cdots, f_{sk}\} \tag{3.2}$$

of software images (e.g., container images) that define the requested resources for that image (e.g., number of cpus, memory or the bandwidth of the virtual NIC). Hence, a service flavor f_{sk} maps a VDU of type s to a specific software image with a set of required resources (CPU, memory, etc). At runtime, to instantiate a purchased VNF, the Orchestrator determines the number of instances needed for each VDU in the VNF as well as the selected image/flavor for each instance and deploys all images as VDUs that are hosted on containers or virtual machines on the virtual infrastructure.

We consider a deployed VNF that is comprised of I VDUs. At any time instance, t_k, let $J_i(t_k) \leq J$ be the number of instances (VDU)s i has $J_i(t_k) \leq J$ running VDUs (instances) with J representing the maximum number of allowable instances of a VDU. Formally, a VNF can be modelled using a graph $G_k(V((t_k), E((t_k))$; a vertex v_{ij} with $i \in \{i, \cdots, I\}$

and $j \in \{1, \cdots, J_i(t_k)\}$, represents VDU j of VDU i. We use nodes $v_{i0} \in V$ to represent the overall behaviour of a VDU i, $i \in 1, \cdots, I$ which as will be shown later will be used to store a summary of the KPIs of all its VDUs v_{ij}. An edge $e(v_{ij}, v_{i\prime j\prime})$, $i \neq i\prime$ represents a virtual communication link between two VDUs v_{ij} and $v_{i\prime j\prime}$, with the edge $e(v_{i0}, v_{i\prime 0})$ modelling the aggregated traffic between all the VDUs i and $i\prime$.As shown in Figure

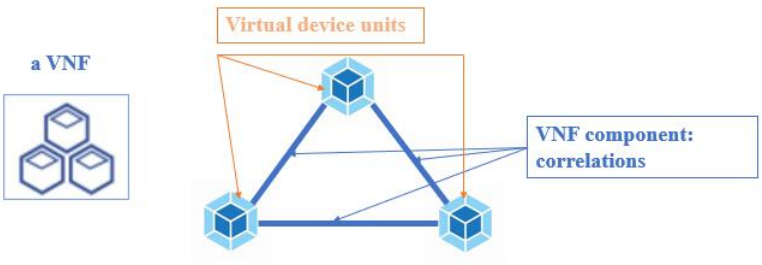

Figure 3.2: An example of a VNF graph with VDU instances

VDUs run on containers hosted on the infrastructure VMs.We represent the virtual infrastructure using the graph $G(N, L)$, where $n \in N$ and $l \in L$ represent a VM and a virtual link, respectively, in the infrastructure. Define the matrices $\Lambda(t_k) = [\lambda_{ij}(t_k)]$ and $\mathbf{B}(t_k)$=$[\beta_{ij}(t_k)]$ of size $I \times J$ such that $\lambda_{ij} \in N$ represents the location of the container running VDU v_{ij} and $\beta_{ij}(t_k) \in s$ indicates the flavor of a VDU v_{ij} which is of type f_s (e.g., a meter or a firewall).

Finally, using various monitoring tools (e.g., [78]) to measure the behaviour of the VDUs, we collect the normalized monitoring information in a third order tensor $\mathbf{Z}(t_k)$= $[\zeta_{ij}(t_k)]$, where $\zeta_{ij}(t_k)$=$(\zeta_{ij}^1(t_k), \cdots, \zeta_{ij}^Z)$. $\mathbf{Z}$ represents performance measurements, such as the throughput, packet processing time and CPU and memory utilization. We also let ζ_{i0} represent higher level aggregated performance for VDU i. We can now describe the state space that describes a VNF at t_k using s_k= $(G_k, \Lambda(t_k), \mathbf{B}(t_k), \mathbf{Z}(t_k))$.

An example from ETSI would explain the correlations in a NS and its underlying VNFFGs. In Fig 3.3, $VNFFG1 : NFP1$ represents a VNF forwarding graph with a network function parallelism mechanism inside, indicating a network service as the subset of the end to end service formed by VNFs and associated virtual links instantiated on the

NFVI as shown from the connection point$cp11$ of the firewall to the connection point of DPI $cp21$, then through $cp31$ and $cp33$ in the NGINX server and connect to the end point of the service function chain.

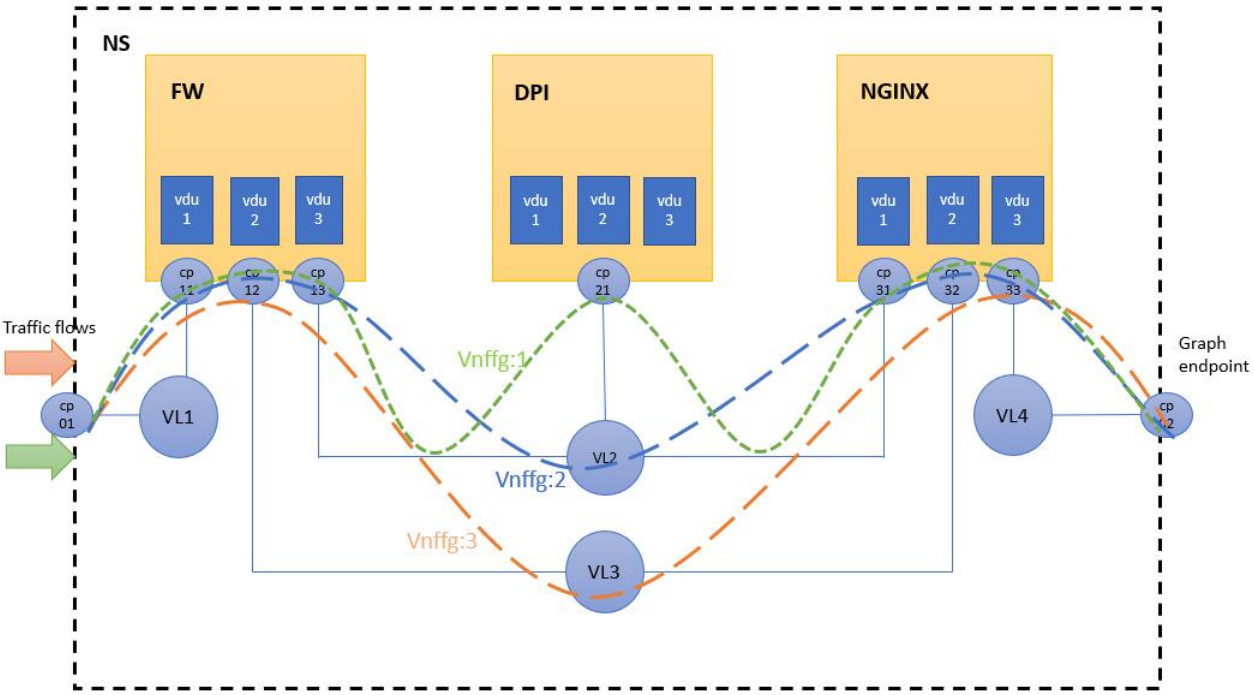

Figure 3.3: A network service with two VNFFGs with diffenret NFPs

3.3.1 VNF states model

State transitions of the modeled VNF can be attributed to either internal or external events. The former may represent a new action taken by the orchestrator (e.g., scaling, migration or termination of a VDU) as well as other monitored events such as an internal software fault. On the other hand, external events include changes in the VNF workloads or in the availability of the consumed resources within the hosting infrastructure. When any of these events takes place at time t_{k+1}, the VNF state changes to s_{k+1}= $(G_{k+1}, \Lambda(t_{k+1}), \mathbf{B}(t_{k+1}), \mathbf{Z}(t_{k+1}))$.

We assume that the OSS/BSS defines an objective optimal performance $\mathbf{Z}^{\star}$ and how costly it is for a VNF performance to deviate from that described value.

The VNFM functionalities are represented as a set of parameterized Action space as follows; we introduce $\mathcal{A}_d = \{a_1, a_2. \cdots, a_{|\mathcal{A}|}\}$ to denote the set of discrete actions that can be performed by the orchestrator on one of the VDU instances of the VNF (e.g., migration, vertical scaling, horizontal scaling, termination, restart).

We associate with each action $a \in \mathcal{A}_d$, a set of continuous parameters $x \in X_a$. For example, an action for vertical scaling would be represented with the (a, x) where action $a=$ *vertical scale* and x is a vector of new values for entries in $\beta(t_{k+1})_{ij}$ in $B(t_{k+1})$.

3.3.2 Metrics observations

Finally, a reward will be given each time the agent maintains the performance closer to the desired optimal value or range as follows. The cost of performance degradation or reward of enhanced performance is measured by a function $g_p(s_k, a_t)$. One example of g_p can be as follows:

$$g_p(s_k, a_k) = \sum_{i=1}^{I} \sum_{j=0}^{J} \sum_{z=1}^{Z} \omega_z (\zeta_{ij}(t_k)^z - \zeta_{ij}^{*z}) \tag{3.3}$$

with $\sum_{z=1}^{Z} \omega_z = 1$. The above function simply calculates a weighted sum of the deviation of the current VNF performance from the desired one.

We also consider the cost of updating the VDU's flavor using the function $g_f(s_k, a_k)$ which reflects the source reduction gains of the cost of the additional resources needed to execute action a_k, that moves s_k with VDU flavors $\Lambda(t_k)$ to $\Lambda(t_{k+1})$. Finally, we take into account the cost of performing the operation itself in terms of the service disruption time $g_d(s_k, a_k)$. This cost can, for example, reflect the downtime as a VDU is restarted or the time it takes to migrate a VDU instance. Hence, the final reward is calculated as a weighted sum of the above reward/cost functions, such that:

$$r(s_k, a_k) = \alpha_p g_p(s_k, a_k) + \alpha_f g_f(s_k, a_k) + \alpha_d g_d(s_k, a_k) \tag{3.4}$$

such that $\alpha_p + \alpha_f + \alpha_d = 1$. It is worth noting that the above PAMDP-based VNF model provides a holistic and automated means to select the appropriate LCM action along

with the required configuration parameters. These actions include creating a new VDU or terminating existing instances where in this case, the configuration parameters would be modified vector representing the graph G_{k+1} and new values for $\Lambda(t_k)$ are created, or existing ones are terminated are calculated in the same manner.

3.4 VNF monitor configuration

In this section, we introduce the metrics we monitored on the VNFs. In general cases, we monitor the necessary metrics in a VDU instance, such as cpu utilization and memory consumption, as well as the throughput, response latency, and working status (to investigate the message sent from the VNF itself, i.e., CLI message) in a fixed time interval T_0. We use a multi-element tuple to represent the state $\mathcal{S}$

$$\int = (cpu, mem, thpt, latency, isvalid) \tag{3.5}$$

In each T_0, we collect the state vector and push it to the reinforcement learning agent for further calculations. After a parameterized action is selected, it will be applied to the VNF, and we observe the new state $\int'$. To evaluate our RL agent, then we use the reward function 3.4 in Chapter 3 to calculate a normalized reward for each action. This reward will be stored in the RL agent to determine the next action. The process will be continuously executed until it reaches the episode upper bound $\mathcal{N}$ or reaches the desired VNF status, which means the metrics will achieve a desired set with acceptable utilization and consumption.

A MDP can be used to represent the VNF state transition. Fig 3.4 shows the states transition with different VNF-LCM operations and the reward based on the monitored measurable states. This formulates a MDP and we can apply RL in our proposed framework.

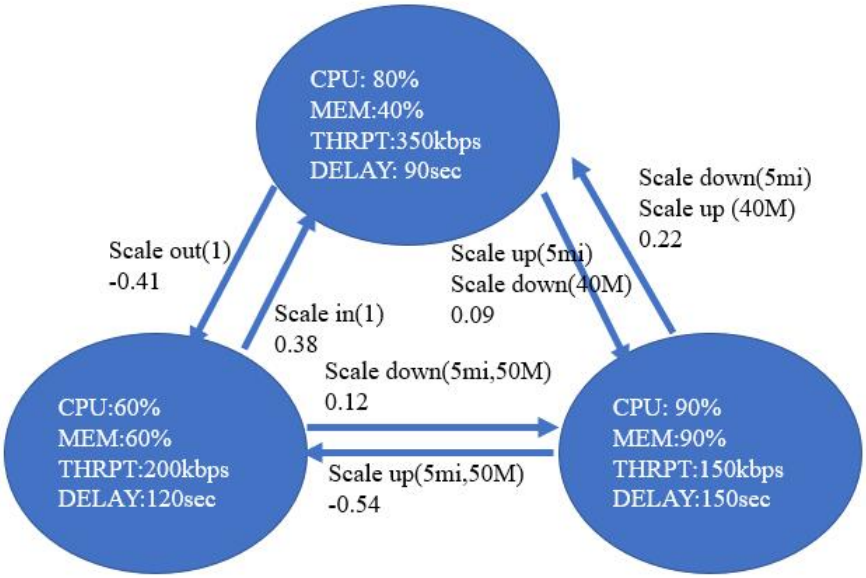

Figure 3.4: MDP example of state transition in VDU1 in a VNF

3.5 Action space

In this section, action space is described, and for each action, a statement will be given. In our work, we design a VNF manager that performs VNF LCM operations, including scaling operations, migration, and self-resilience (reboot). In the latter subsections, we will introduce all the actions we designed.

3.5.1 Scale-up operation

A scale-up operation is an essential scale operation that directly increases the current resource consumption upper bound to provide more available resources for the VNF instance.

Typically, a scale-up operation includes the type of resources to increase (i.e., assign more cpu resources to the VNF) as well as the specific amount of resource adjusted:

$$a_{scale-up} = (resource - type, resource - amount) \tag{3.6}$$

We can use a figure to represent a scale-up operation: In this case, the CPU usage upper

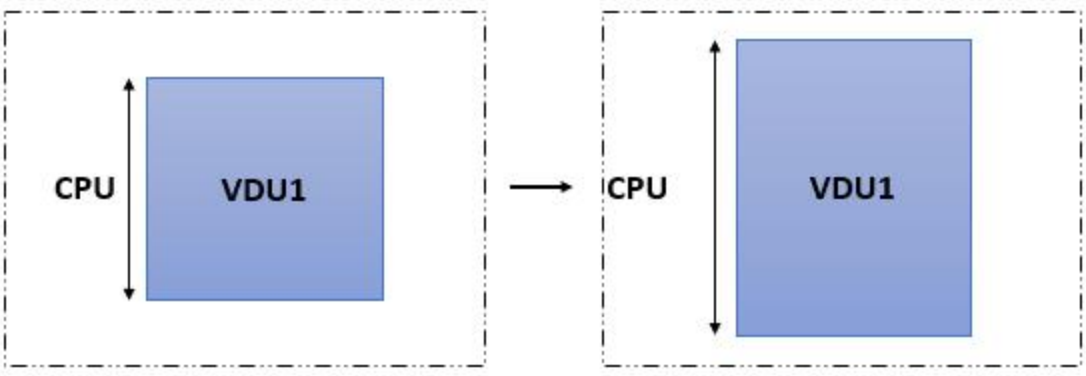

Figure 3.5: A scale-up operation

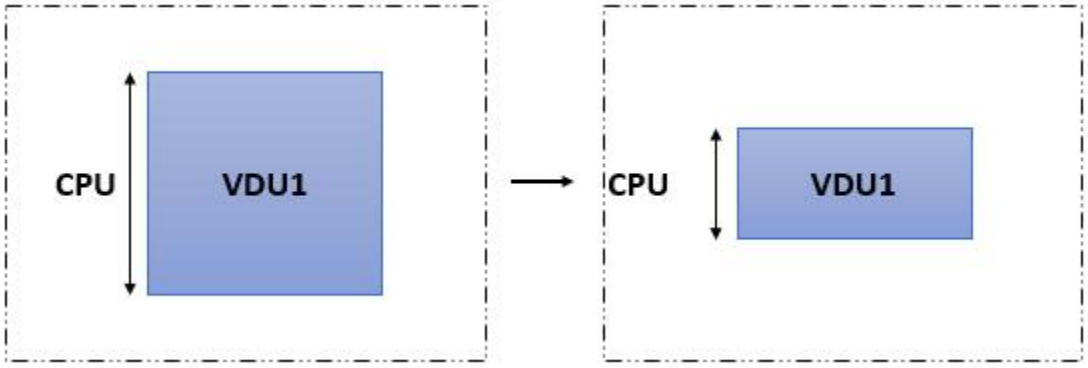

Figure 3.6: A scale-down operation

bound is increased while the memory consumption limit remains the same value.

3.5.2 Scale-down operation

By contrast, scale-down executes in the opposite direction to decrease the number of available resources. Similarly, a scale-down operation includes the type of resource to adjust as well as the amount of resource that scaled:

$$a_{scale-down} = (resource - type, resource - amount) \tag{3.7}$$

We can use a figure to represent a scale-up operation: In this case, the CPU usage upper bound drops while the memory consumption limit remains the same.

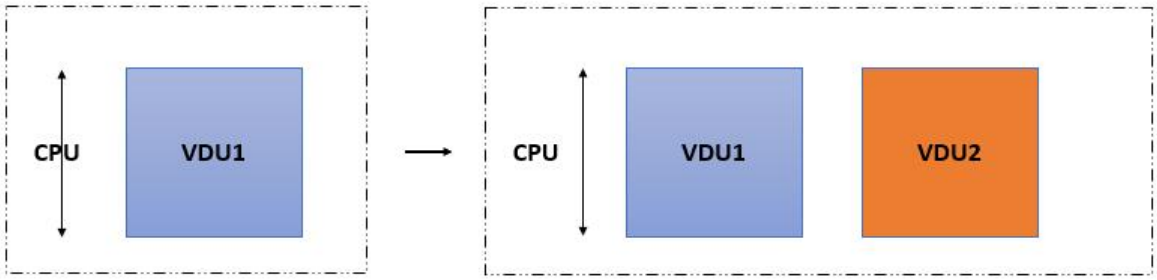

Figure 3.7: A scale-out operation

3.5.3 Scale-out operation

We introduce horizontal scaling in the following subsections. Horizontal scaling refers to operations related to VNF instances themselves (i.e., add up the number of instances) while not overwriting the VNF configuration (i.e., changing the resource allocation configuration inside a VDU).

A scale-out operation is a horizontal scaling process that adds up the number of VDUs inside a VNF. To execute this operation, we need to determine how many instances are required to add in the VNF. In general cases, the more instances number we add, the more complicated the configuration work will be:

$$a_{scale-out} = (instance - number) \tag{3.8}$$

We define the action as a integer with the number of instance we added to the VNF.

3.5.4 Scale-in operation

A scale-in operation has the similar format of the scale-out one, where it oppositely rewrite the number of VDU instances:

$$a_{scale-out} = (instance - number) \tag{3.9}$$

Similarly, the action is defined as a integer with the number of instance needed to decrease.

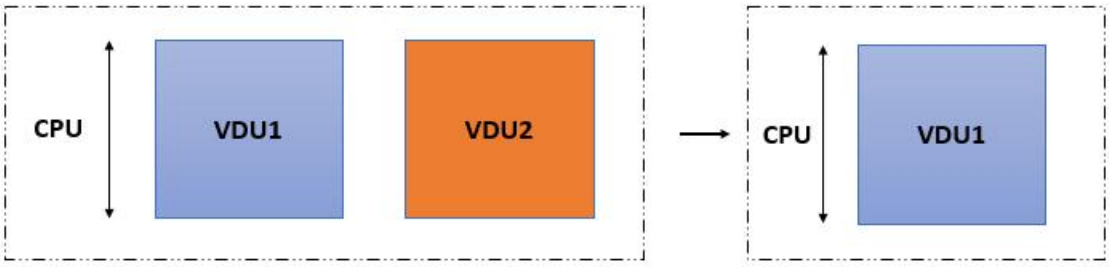

Figure 3.8: A scale-in operation

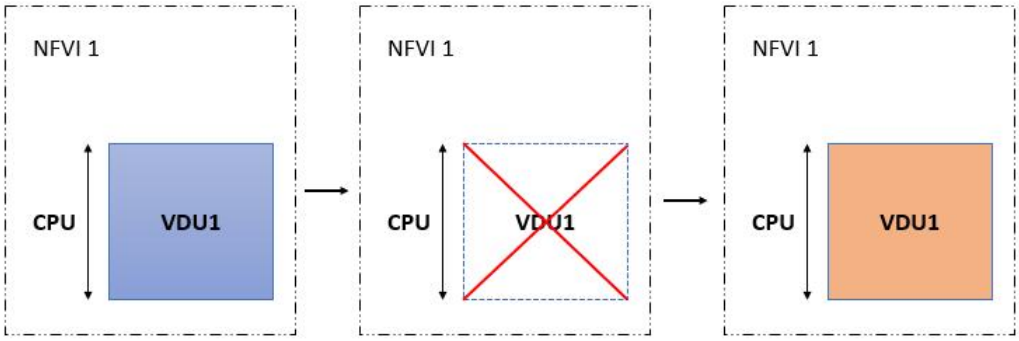

Figure 3.9: A reboot operation

3.5.5 Reboot operation

The previous scale operations cannot completely fulfill the requirement of VNF-LCM operations. When VNF failure happens, which means the VNF currently is not working correctly, those scale operations may not bring the VNF back to regular status. Typical VNF failure includes VNF crash and link failure. A VNF crash always comes with an error message from the VNF itself, while the link failure needs detection techniques. Here we use a reboot to overcome the potential problems: A reboot operation firstly requires the VIM to kill the current instance as well as the corresponded configuration(connection points, virtual links), which brings a VNF downtime in case that there is only one VDU or increases the burden of the other VDUs.Then a new configured VDU with the exact same configuration is instantiated and its newly updated configuration. This operation is much time-friendly than the scaling operation since the VNF manager does not have to modify the configuration. However, the performance can be bad and unstable. This operation is considered an optional VNF LCM operation in most cases.

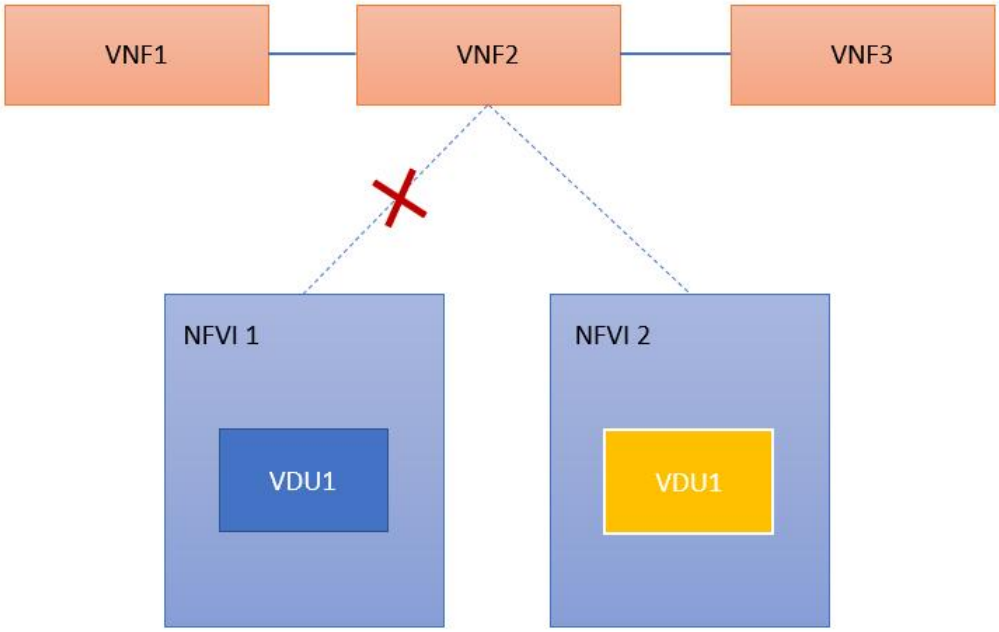

Figure 3.10: A migration operation

3.5.6 Migration operation

As we defined in Equation 3.1. One NFV infrastructure is limited with a resource pool with a total available resource upper bound (i.e., In a datacenter cluster, there is the largest number of available computation resources). Therefore, in some cases, when Equation 3.1 cannot be satisfied, a migration operation will be triggered. In other cases, such as when there is a VNF failure, migration operation would perform differently from a normal scaling operation: The example shows VDU1 of VNF1 migrate from one infrastructure to another. Similar to scaling operations, a migration operation needs re-configuration and takes a lot of state transitions. In general cases, the response latency of a migration operation is much higher than the scaling one.

3.5.7 Potential additional actions

In our work, even though we extend the action space to a larger space. However, there are potential additional actions that can be considered in future work. In our proposed model, we consider a distributed NFVI with two possible nodes. Obviously, we cannot

apply any parameter to the migrate operation since there is only one alternative physical node for the VNF. If the NFVI is based on a more distributed case, for example, edge servers, it is likely that our algorithm might cause a problem because of the randomly assigned instance (in migrate operation). As a result, if we consider a more distributed NFVI, it is important to make a VNF-placement related parameter selection mechanism in the action space. A MDP can be used to represent the VNF state transition. Fig 3.4 shows the states transition with different VNF-LCM operations and the reward based on the monitored measurable states. This formulates a MDP and we can apply RL in our proposed framework.

3.6 RL-based action selection

In this section, we will introduce the proposed algorithm. The VNFM selects actions based on a learned policy that aims at maintaining the performance of the VNF as close as possible to the target performance $\mathbf{Z}^{\star}$. Following Wei et al. [79], we define a compound policy for the selection of the parameterized actions as follows.

$$\pi(a, x|s) = \pi(a|s) \times \pi(x|s) \tag{3.10}$$

where π is the compound policy, a is a discrete action, x is the parameter bound with the action. $\pi(a|s)$ is the discrete action sampled from a parameter x from $\pi(x, s)$. One possible solution to realizing the desired policy is to flatten the decision-making. This can be achieved by merging the action and parameter sets to construct a very large action space. Two main limitations can be identified in this approach. The first is the huge action space. Additionally, we lose the fine-grained tuning of the parameters of individual actions. We calculate optimal discrete policies using value-function learning while the action parameters are calculated using policy gradient methods, which means we apply reinforcement learning algorithms twice in our case. With this mechanism, the output size of the parameter policy will be greatly reduced.

Define a function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. Here, $Q(s, \mathbf{a})$ represents the expected-long term

reward obtained by taking an action a in a given state s.

$$Q(s,a) = \mathbb{E}_{r_t, s_{t+1}}[r_t + \gamma \max_{a\prime} \in \mathcal{A} Q(s_{t+1}, a\prime) | s_t = s, a_t = a] \tag{3.11}$$

where $\mathbb{E}$ means taking an expectation over all possible states and rewards. r_t is the reward gained at t after taking action a.

Algorithm 1 defines the main steps needed to calculated our optimal action-parameter policies. We first assign arbitrary values to the state-action pair in the Q table. Then we update the action parameters for all the actions, Θ, using Algorithm 2. The initial action is then chosen based on the initial Q table, and then the reward function will ensure those VNFs have distinguished monitoring metrics. Next, we apply an epsilon-greedy-based deterministic method to choose the best state-action pair from the Q table. Once again, Algorithm 2 is called to update the parameters Θ. We repeat this work until the Q table is fully updated in the episode.

Each time a discrete action is selected, Algorithm 2 is called to determine the corresponding parameters for the selected actions. Note that this algorithm can be changed to suit the nature of different discrete actions. More precisely, other existing ML schemes for placement, migration, and scaling, for example, can be used to replace that algorithm. In general, Algorithm 2 employs a policy gradient-based algorithm to calculate the action-parameter. More precisely, as we already have trained a good model for obtaining discrete actions, we then train the action-parameter x. We set up the discrete action based on the Q-table Obtained in Algorithm 1. We then initiate a resource parameter arbitrarily in a constant set range. After executing the action, we observe the state transition and the reward and update the parameter network θ to find the maximum expectation of the value function θ^* using the following equation:

$$J(\theta) = \mathbb{E}(r_1 + \gamma r_2 + \gamma^2 r_3 + ... + |\pi((a,x),\theta)) \tag{3.12}$$

Where γ is the attenuation factor and $(r_1, r_2, ...)$ is the reward in each episode.We use a

Monte-Carlo based REINFORCE algorithm [80], to update θ :

$$\theta^* = \theta + \alpha \nabla \log \pi_\theta(s_t, a_x) v_t \tag{3.13}$$

Where v_t is the value function obtained in the episode, in each episode, α is the learning ratio,a_x is the parameterized action. The algorithm will update the action parameter, and at last, we will return the action parameter network set Θ.

3.6.1 Discrete action selection

We introduce the discrete action selection algorithm in the following text.

Algorithm 1 Discrete action selection

Initialization:

Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}$, $a \in \mathcal{A}$.

$\Theta \leftarrow$ UpdateParameter()
while Q-table is not full **do**
  $s \leftarrow$ current state
  $a \leftarrow \pi(s)$
  Take action a, observe reward r and new state $s\prime$
  Choose a' from s' using policy derived from Q
  $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$
  $\pi \leftarrow \epsilon$-greedy w.r.t.Q.
  $s \leftarrow s', a \leftarrow a'$
  $\theta \leftarrow$ UpdateParameter()
  if Q table is fully updated in the episode **then**
    jump out of iteration
  end if
end while
return Q (s,a)

3.6.2 Action parameter selection

We present our action-parameter selection algorithm in the following page. Figure 3.11, presents an example of our system working to execute a scale-in operation in a given service function chain. When the overall CPU utilization is high, the agent notices the abnormal behavior observed from the VDU instances. Then, the agent triggered an action by connecting the VNFM and applying it with the proposed algorithm's parameter. For example, a $scale - in$ action includes the name of the VDU instance, the discrete action

Algorithm 2 UpdateParameter

Initialization: current state of the VNF

Initiate action parameter train network θ
Initiate action parameter x
Initiate action parameter network set Θ=[ ]
while discrete actions set is not iterated **do**
  choose discrete action a from $Q(s, a)$
  while episode number t in range T **do**
    execute parameterized action (a, x)
    observe s_t and get reward r_t
    $\theta^* \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_x) r_t$
  end while
  push (a, θ) to Θ
end while
return Θ

and its correlated parameter $VDU_{count} = 1$, and the VNF type parameter *normal*. After the action is executed, the system uses the monitor to observe the VNF status, calculate the reward based on the variance over the status, update the Q-table and the parameter network.

3.7 Summary

In this chapter, we introduced the structure of our VNF and its component. To implemented a good VNF model, we also introduced the NFV infrastructure requirement and our monitor objectives. We then discussed our hypervisor insights and VNF state model to convert the problem to a VNF status optimizing problem. We presented our VNF monitor configuration afterward, including the metric we observe, and review the state space we created. Then we discussed the action space, covering all the actions in detail on its functionalities, advantages, and disadvantages. We lastly presented the RL-based algorithms,

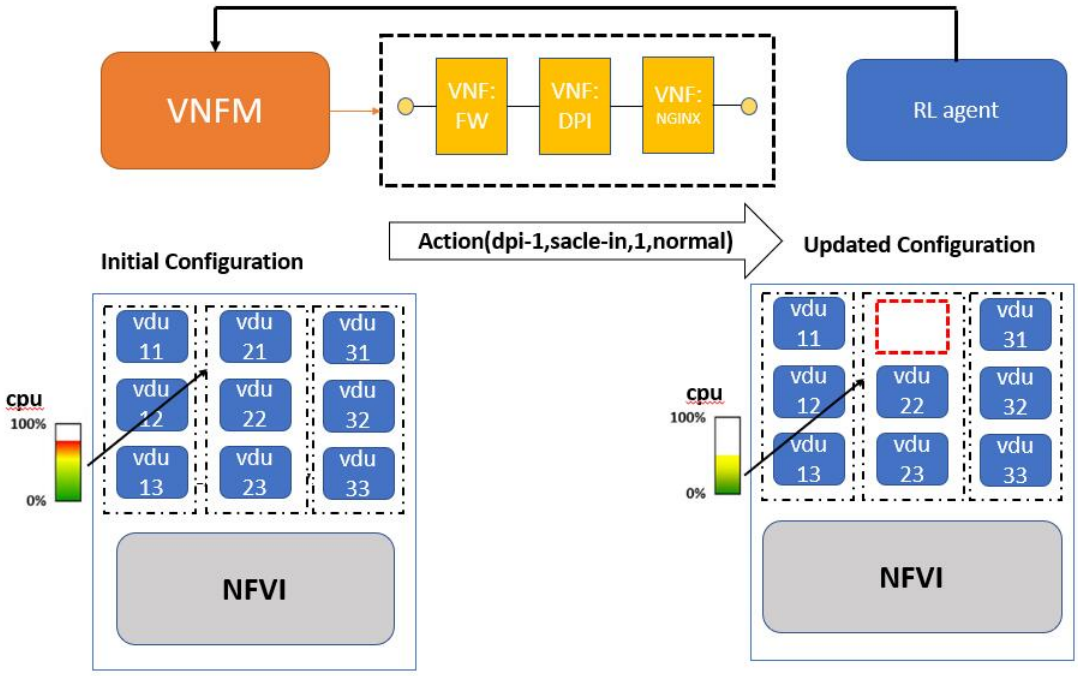

Figure 3.11: Example: scale-in of DPI-VNF

including a first phase, discrete action selection, to determine a discrete action with a limited action space along with an initial parameter using Q-learning; then, we trained the action-parameter in the action-parameter selection algorithm using Policy Gradient policy update method. In the next chapter, we will present the performance evaluation based on our current algorithm and apply it to the model we derived.

Chapter 4

Performance Evaluation

In this chapter, we set up the experimental environment and use PYTHON to test and evaluate our proposed framework and algorithm. We use Kubernetes cluster to represent the NFV infrastructure, Kuberenetes pods to represent the VNF and containers to represent virtual device units. Then we use PYTHON language to construct a two-step RL algorithm to determine the performance. Section 4.1 gives the experiment set up insights. Section 4.2 presents the performance evaluation. Section 4.3 summarizes the chapter.

4.1 Experiment set up insights

To evaluate the performance of our proposed VNFM , we first design the experiments with two cluster nodes with Intel(R)Xeon 1.90GHz cpu and 32GB of RAM and a jumpserver working as a master node in Fig 4.1. We used Kubernetes to build our simulation environment where we modelled the behaviour of a single VNF that is comprised of two VNDUs. We initially set up the same resource limit and resource request, respectively, for each VDU as follows:("cpu":"200m","memory":"250Mi"), ("cpu":"80m","memory":"120Mi"). We built a both full-connected two layer network for the DRL algorithm, the number of neurons is initially set up as (1000,61), where 1000 is the number of states and 30 is the possible action parameter values which varies from -30 to 30 (the step length for kubernetes pods configuration updates are integers). In the experiment, we employed the cpu

utilization and memory consumption as the two core metrics that we monitored using fixed time intervals T_0 (90 seconds). We compare our VNFM functionality against a static configuration of Kubernetes default resource scheduling and allocation schemes. In Table 4.1 we can find the parameter set-up.

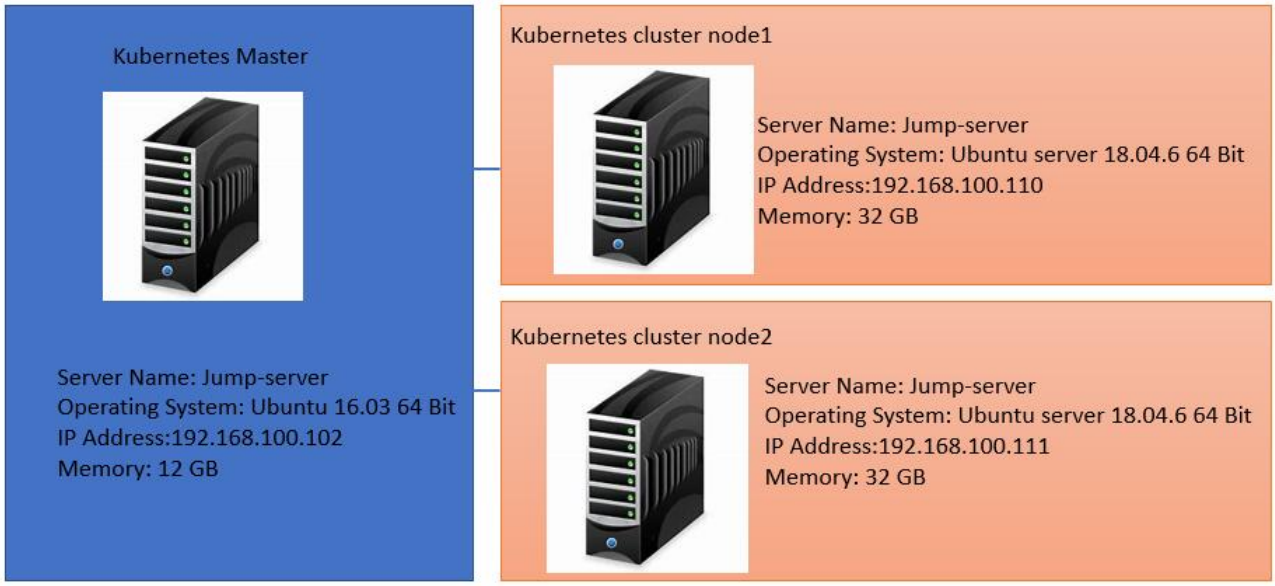

Figure 4.1: Kubernetes configuration

Table 4.1: Simulation parameters setup

Parameter	**Value**
$vdu_{cpu-limit}$	$200m$
$vdu_{mem-limit}$	$250Mi$
$vdu_{cpu-request}$	$80m$
$vdu_{mem-request}$	$120Mi$
N	1000
θ_{step}	$(-30, 30)$
T_0	90 s
ϵ_c	60
ϵ_{greedy}	0.95
α_{ql}	0.3
α_{pg}	0.1
γ	0.95

We designed SFC including three VNFs: a NGINX server with a web application; a virtual firewall (vFW) and a virtual intrusion detection system (vIDS). The flows generated

firstly visits the vFW, then the vIDS and lastly goes to the NGINX server.

4.2 Performance analysis

In this section, we go over the main evaluation experiment and analyze the performance. First of all we analyze the performance of discrete action selection algorithm by observing the Q-value we collected; then we compare the throughput we monitored for each VNF and for each type of VNF LCM operation; meanwhile, we compare the response delay in terms of different VNF-LCM operation. At last we do a summary on this chapter.

4.2.1 Q-table comparison

We first examine the performance of single shot action parameter selection algorithm using two discrete actions, namely, scaling up and scaling down. Figure 4.2 provides the corresponding Q-values for the actions against one of the state parameters which is the cpu utilization. It is worth noting here that we discretize the cpu utilization values it into ten bins each with a width 0.1. As depicted from Figure 4.2, when the cpu utilization increases, the Q-value of $scale-up$ grows dramatically from 0.169 to 0.856 while the Q-value of $scale-down$ decrease from 0.756 to 0.140. This ensures the general direction of our testing work as well as the basic automatic VNFM ability in distinguishing the appropriate actions based on the VNF state. Scale-in and Scale-out operation is in the extension experiment. And a compound comparison is made in the following Figure 4.3. In Figure 4.3 we can find a similar trajectory that the scale-in and scale-down operation has the close effect on the VDU instance when the cpu utilization of the VDU is relatively low, which indicates a large idle time or a less dense workload for the VDU. As a result, the Q-value of scale-in and scale-down has much higher value than the other two operations. Interestingly, scale-in operation has even higher Q-value than the vertical operation when the utilization is at a close-to-zero value, and this shows the VNF manager is more likely to execute a horizontal scaling operation rather than direct resource re-allocation.

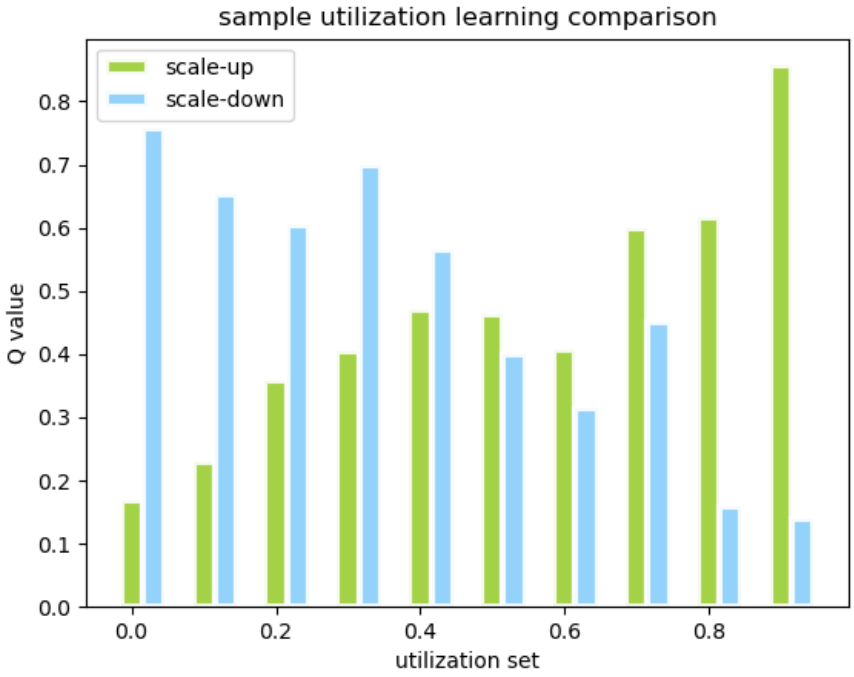

Figure 4.2: Q-value comparison.

On the other hand, when the utilization set is relatively large, indicating a more utilized VDU instance, the Q-value of scale-up and scale-out has a clear advantage over the other two operations. Similar to the previous two operations, the scale-out operation, which is also a horizontal operation, has a higher Q-value than the vertical one.

4.2.2 Action parameter comparison

Next, we examine the performance of our policy gradient based algorithm in assigning accurate action-parameters once the discrete action is selected. In Figure 4.4 we plot the θ values, associated with the two actions of scaling up and down against the VDU state that is reflected by the cpu utilization.

The multiple points reflect different parameter values as the VNFM refines its decisions, and negative values (below zero) mean that the action is not selected. As the cpu utilization decreases from 40% and approaches 0% , the parameter for the scale-down action increases gradually until it approaches around 30 milli cpus. On the other hand, as the cpu utilization

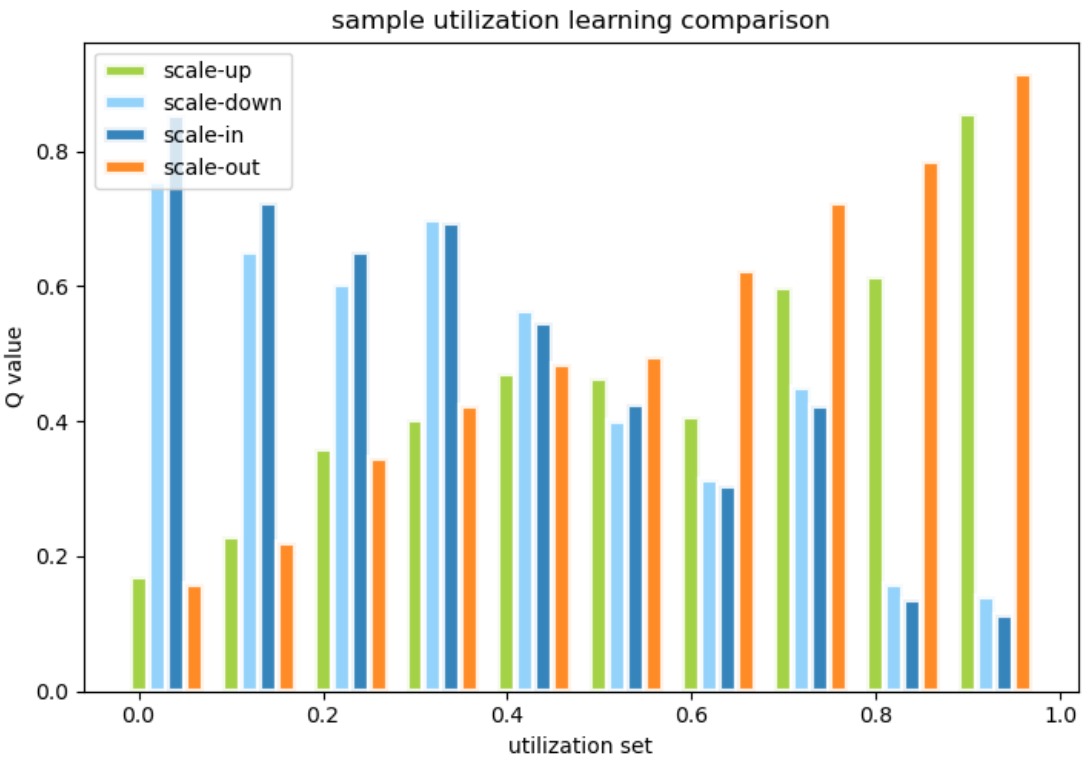

Figure 4.3: Q-value comparison with more actions

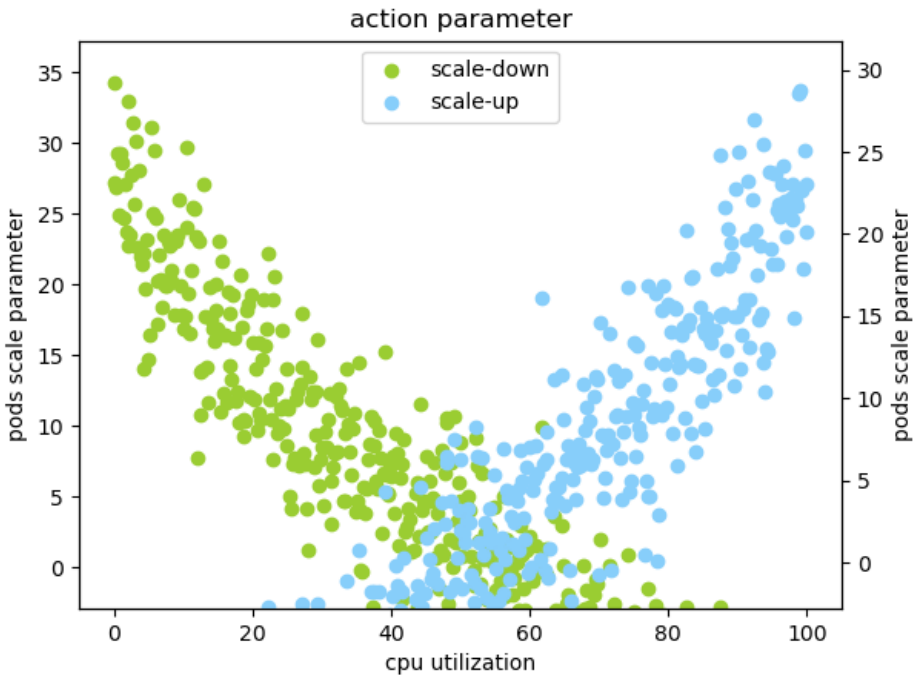

Figure 4.4: Action parameter values.

increases, the parameter of the scale up action increases gradually to allow the VNFM to gracefully increase the allocated resources to the VDU.

4.2.3 Utilization and throughput comparison

In the subsection, we discuss the throughput comparison. First of all we use three scenarios with horizontal scaling operation:

- Scale-out operation with parameter θ=1
- Scale-out operation with parameter θ=2
- Scale-in operation with parameter θ=2

In terms of a scale-out operation, as discussed in Chapter 3, it requires an integer parameter as the number of instance to scale. We tested the two cases with θ=1 and θ=2, respectively. Figure 4.5 shows a scale-out scenario for a VNF with one VDU initially. The traffic is

increased within the time. The figure shows a gradual increase for VDU1's CPU utilization, where it triggers a scale-out operation at a timestamp around 12. At this timestamp, a duplicated VDU (VDU2) is instantiated, and traffic is redirected to VDU2. We generated a service label for all pods, which ensures the traffic goes to the external service port and redirects to the pods, respectively. This guarantees a load-balancing mechanism, and at timestamp 25, it can be seen that the CPU utilization of both VDUs remains the same. Figure 4.6 shows a more complex scenario with a scale-out parameter θ=2. With a similar mechanism, we increase the traffic and triggers a scale-out operation. The figure shows that after the scale-out operation, the CPU utilization for all the VDUs comes to a stable stage at around 0.6, where the scaling the utilization has arisen to 1.0 for VDU1. The scale-in operation has an opposite effect on the VDU instance. As it shows in Figure 4.7. Initially, the CPU utilization of all VDUs is at approximately 0.6. As the traffic drops to a certain point, the workflow for all the VDUs has decreased within time. As a result, a scale-in operation is triggered, and 2 VDU instances were killed to release the resources. It

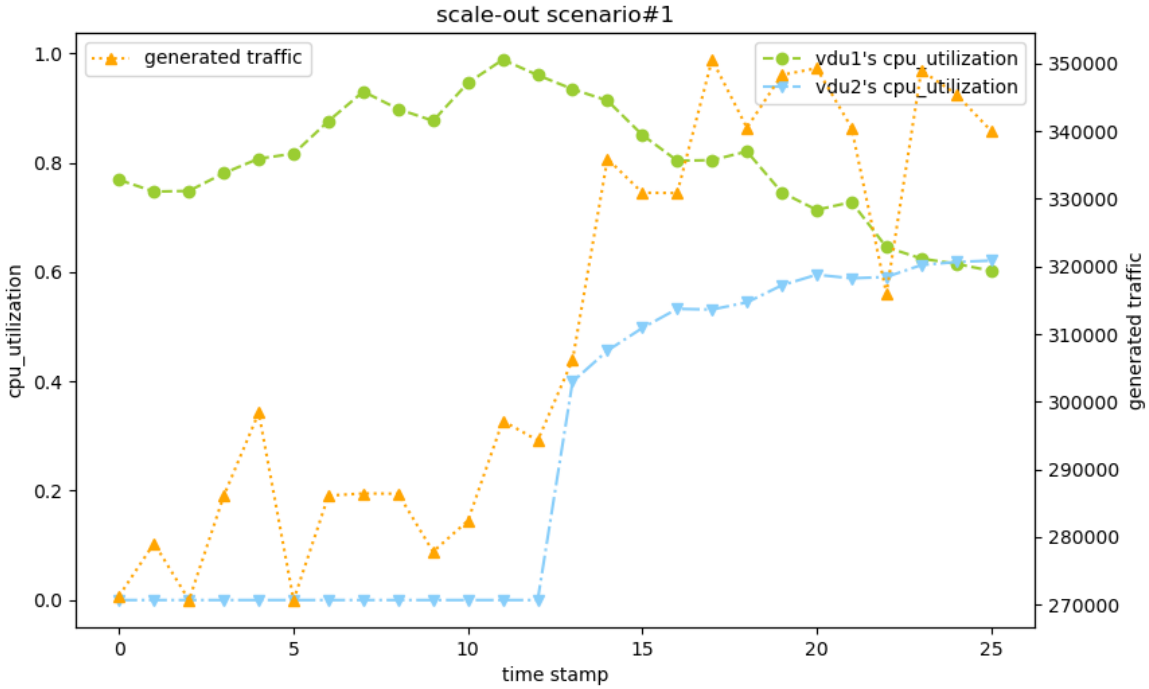

Figure 4.5: Scale-out operation with θ=1

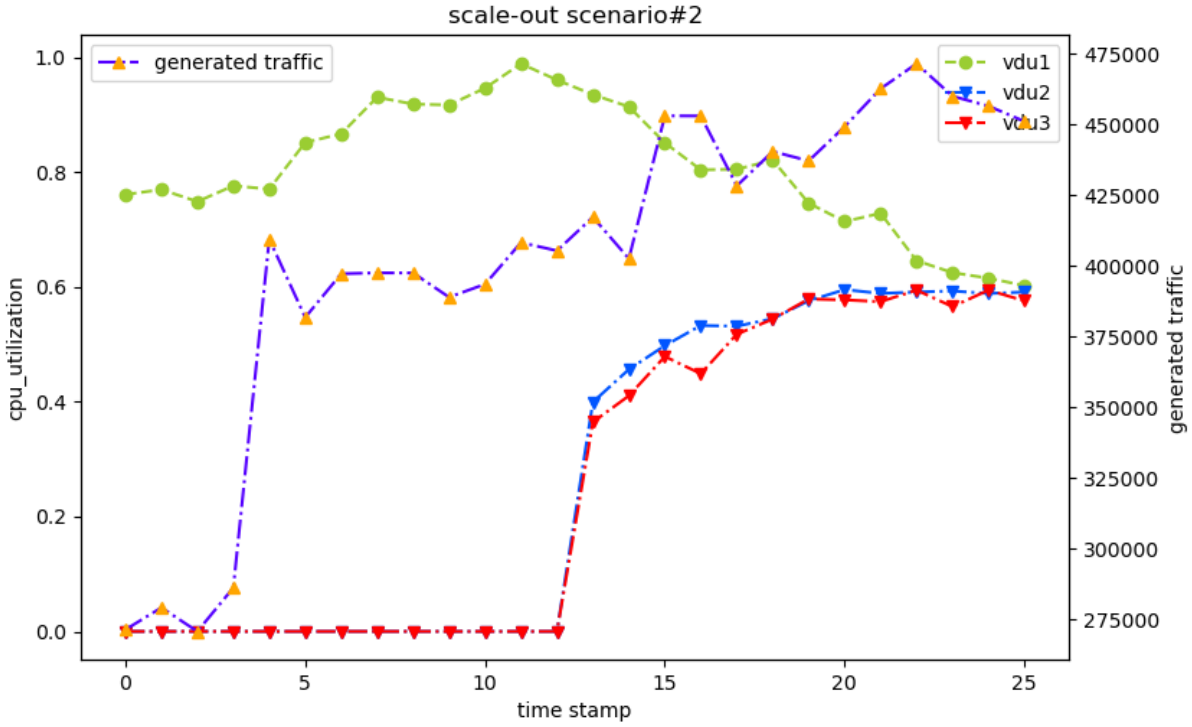

Figure 4.6: Scale-out operation with θ=2

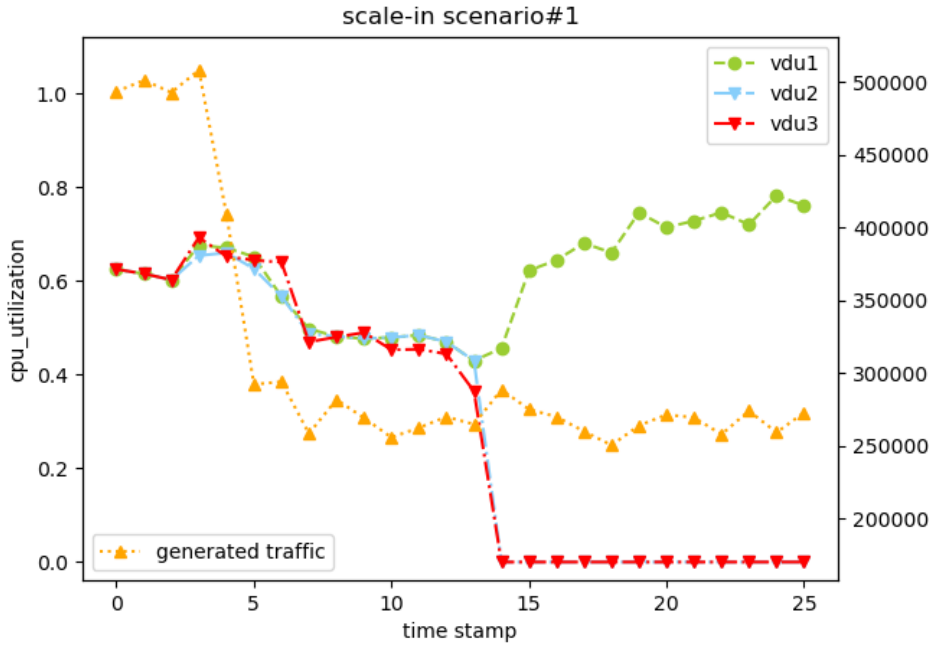

Figure 4.7: Scale-in operation with θ=2

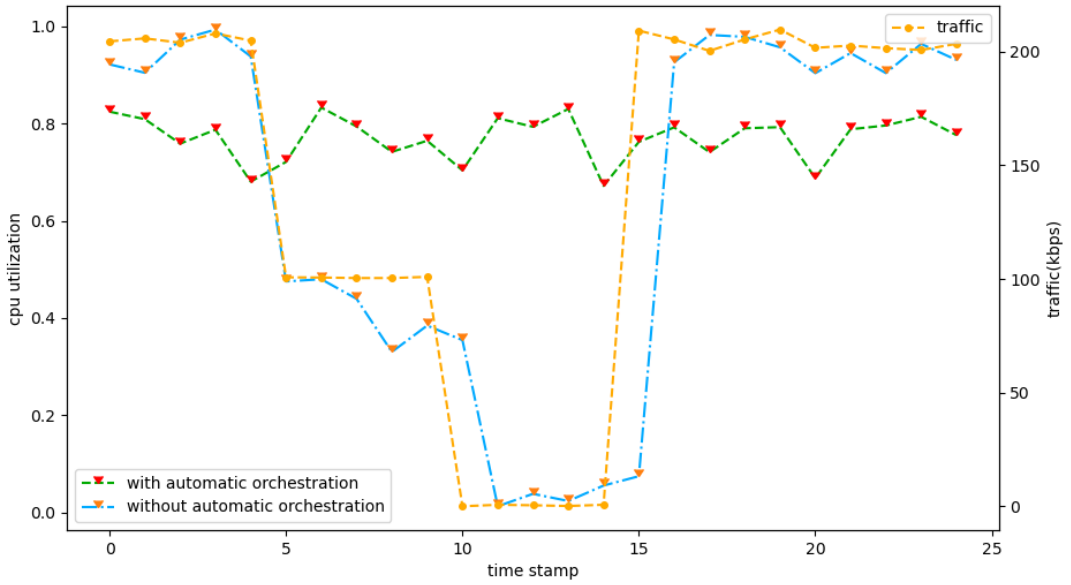

Figure 4.8: Average CPU utilization for the NGINX server

can be inferred that if a VNF has multiple VDUs, it is more likely to execute a horizontal scaling operation in terms of sudden throughput changes.

Another comparison is the overall throughput comparison within our tested model and the plain service function chain without any VNF-LCM operations. In Figure 4.8, it shows an example of the CPU utilization of the VNF NGINX server. By comparison, it is easy to find the model we promoted ensures a relatively stabled state of the VNF, while without any VNF-LCM operations, the CPU utilization can vary within the time and traffic changes, which may bring unstable state to the VNF itself as well as the network service. This is what we desired for the VNF-LCM mechanism to bring a stable environment to the entity of the virtualized architecture with respect to resource and security concerns. Figure 4.9 represents the overall throughput for the service function chain. As the figure shows, the proposed model improves the efficiency of resource utilization. In return, the overall throughput is higher than the one without any orchestration framework.

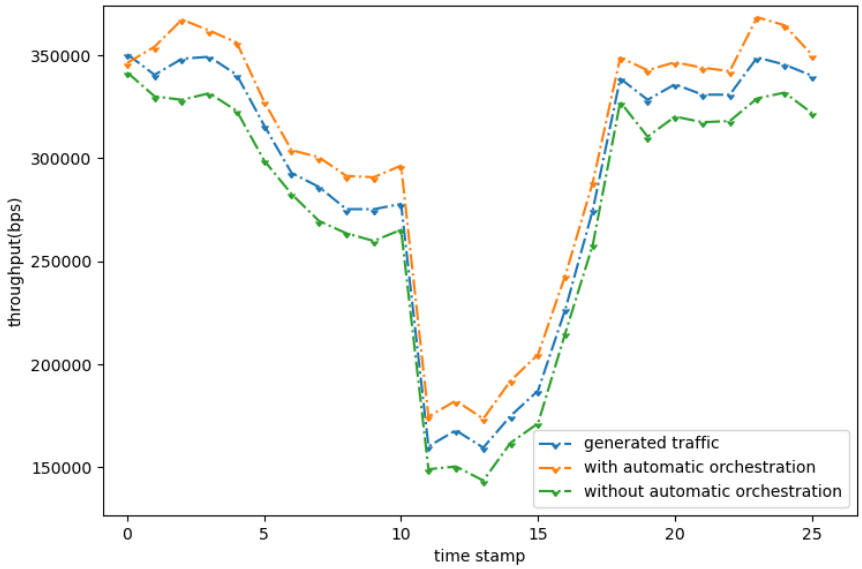

Figure 4.9: Throughput comparison

4.2.4 Response delay comparison

In this subsection we will use the delay comparison to evaluate our work. Firstly we would like to discuss the delay for each VNF-LCM operation. As introduced in Chapter 4, different VNF-LCM operations may lead to different response times. Therefore, we firstly compared the overall response delay between VNF-LCM operations. The figure shows that the scaling operation has a close response time among each other at around 100 seconds. This is because all the VNF operations need state transitions, which means the scaling operations require reconfiguration operations(i.e., modify the pods' configuration file and re-activate the instance). Compared to rebooting operation, it does not need a reconfiguration operation, leading to a shorter downtime for the VNF. The migration operation requires a total remapping from one NFVI to another. This is also the reason why migration takes a much longer time than any other VNF-LCM operation.

In this case, it is not that likely to select migration as a VNF-LCM operation due to its long downtime. However, we studied some cases in which migration has better performance than other VNF-LCM operations. If there are occasions that shutdown the link or the VNF itself, the normal scaling operation might not work appropriately. In

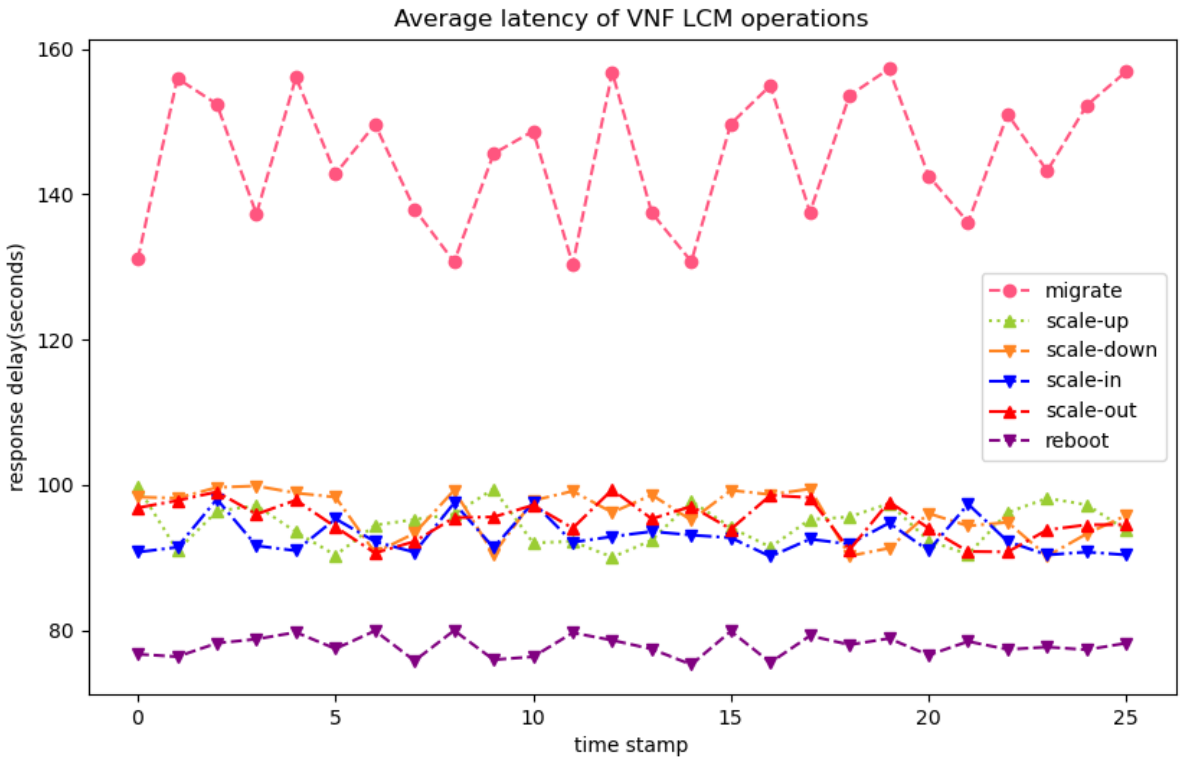

Figure 4.10: Average latency of VNF LCM operations

general cases, since we added the $isValid$ parameter to the states when there is a VNF crash, the VNF will report the error message to the hypervisor (i.e., Kubernetes Master nodes) and VNF manager. It can also be determined by typical throughput changes as shown in Figure 4.11 This figure shows the VNF has 2 VDUs, and at the timestamp of

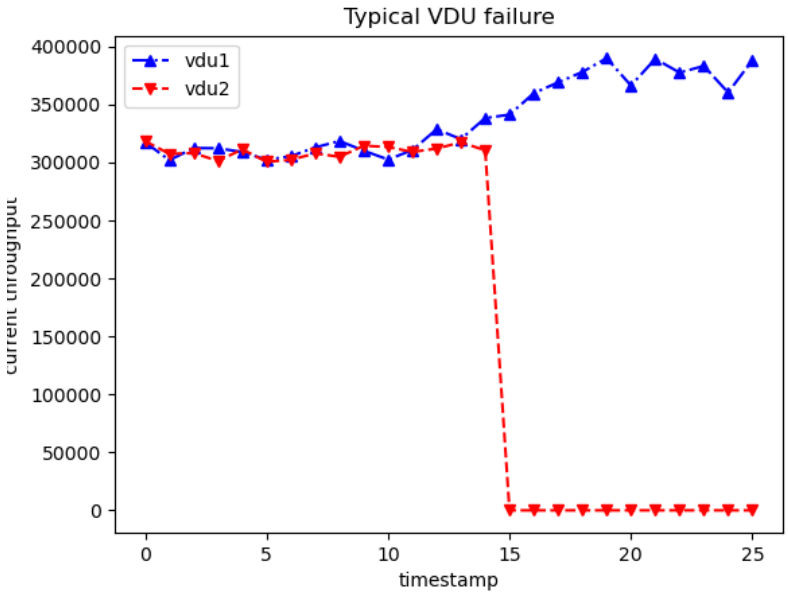

Figure 4.11: Typical VDU failure

about 15, the current throughput of VDU2 dropped to 0 while the current throughput of VDU1 gradually raised. We can infer that VDU2 has encountered a link failure or VNF crash. Let's see what will happen if we perform an action when there is a VNF failure. Figure 4.12 shows an experiment with one VDU failure and see the changes with different VNF-LCM operations. In Figure 4.13, we set the VDU1 as the normal VDU and do not apply any changes to it. We trigger a VNF failure in VDU2 at the timestamp around 11 and observe the results. The blue line represents a reboot operation. It shows after the rebooting operation, the VDU goes back to work with a lower throughput (this is because of the initial set-up configuration process) using the exact previous configuration before the VNF failure. However, the orange line showed the difference between a scale-out operation and a reboot operation. By scale-out the VNF, we added another VDU instance to the VNF. The direct impact is to re-configure the network. As a result, the downtime for scale-out operation is longer than rebooting. Then we tested the migration. As it showed

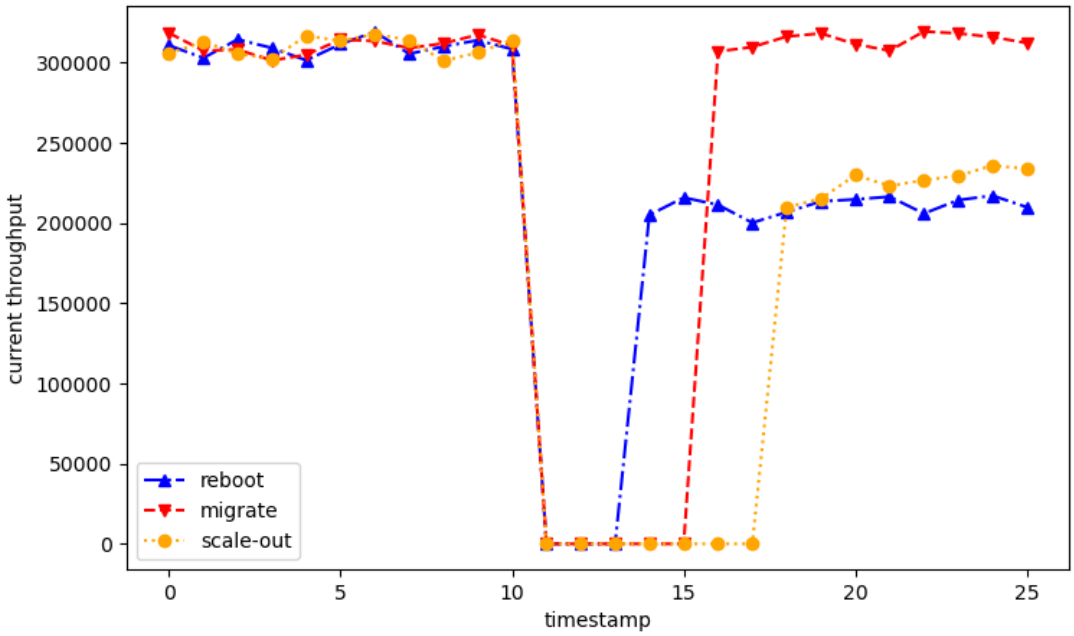

Figure 4.12: Throughput changes with VDU failure

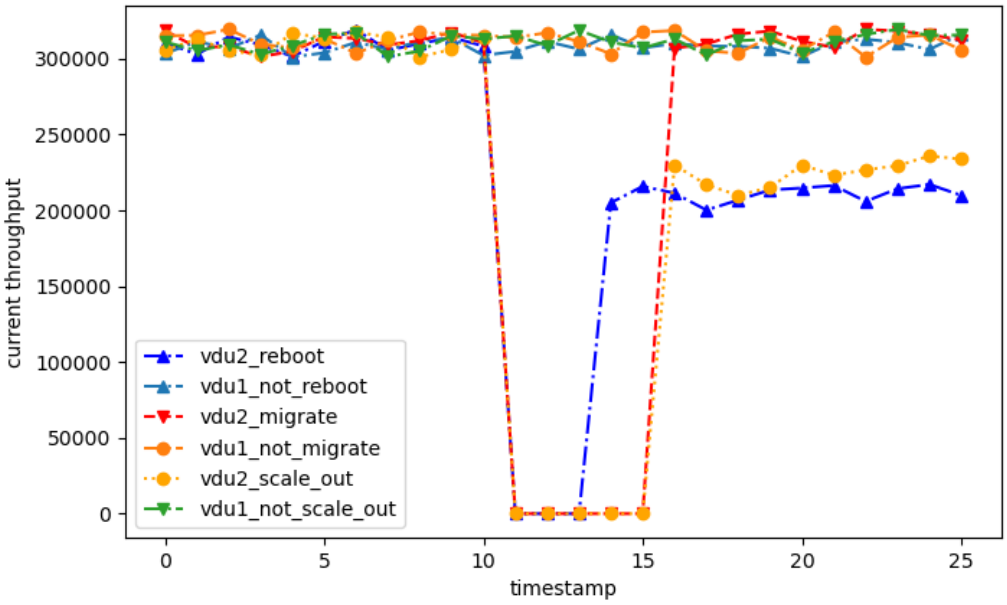

Figure 4.13: Throughput changes with VDU failure, with a comparison of initial VDU

in the figure, the migration has a better performance than rebooting and scale-out. The throughput reaches the previous level at a sacrifice for a more extended down-time. This indicates a VNF crash or link failure. Migration or reboot would have a better performance. Figure 4.14 presents an overall comparison with our proposed model. The throughput of

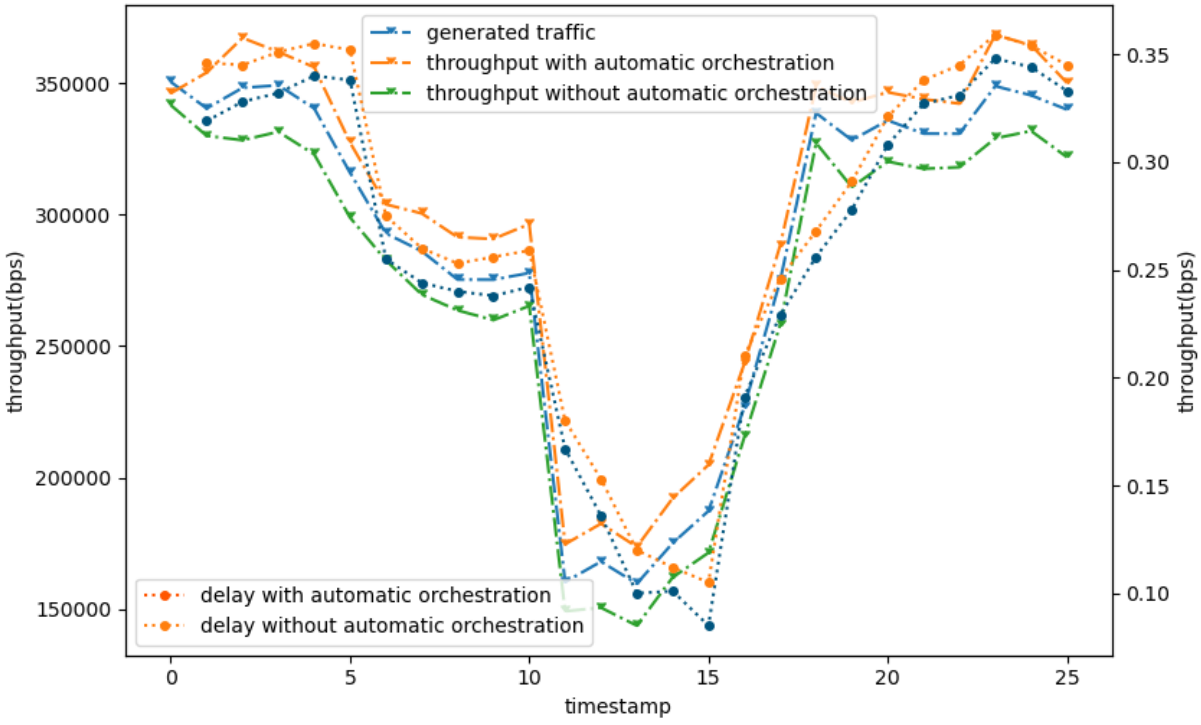

Figure 4.14: Overall delay with the proposed model

both schemes follows the workload pattern as depicted by Fig 4.9. However, our proposed VNFM achieves better throughput. Similarly, the delay experienced by the flows decreases as the workload is decreased. Nonetheless, there is a noticeable improvement in the delay achieved by the proposed VNFM as shown.

4.3 Summary

In this chapter, we firstly introduced the experiment set up parameters. We then analyze the discrete action selection by observing the Q-value derived from the Q-table. Then we present the action parameter derived from the PG algorithm, which showed a remarkable performance in terms of different utilization set. Besides, we evaluated our proposed model by observing the utilization changes and throughput updates with different VNF-LCM operations. Lastly we compared the response delay of those actions and evaluate the overall performance of the proposed system and it can be seen the proposed algorithm can automatically and optimally manage the VNF instance in an ordered manner.

Chapter 5

Conclusion and Future Work

In this chapter, we present the conclusion and future work we would like to study on.

5.1 Conclusion

The management of VNFs has been a hot topic in academics. The proper resource online allocation mechanism can dramatically improve the efficiency of management work as well as the stability of the system. In this thesis, we presented a novel model targeting VNF-LCM management via a compound reinforcement learning algorithm. We formulated the problem of the life cycle management of a single VNF as a Markov decision process (MDP). By comparing the different VNF-LCM operations, we proposed an RL-based action selection mechanism to choose the proper VNF-LCM action to realize a stable MANO system from the aspects of VNF management. One of the main features of the proposed model is the employment of parameterized actions. Existing RL-based solutions are limited in that they either address a single action (e.g., scaling) or non-parameterized action selection (e.g., vertical vs. horizontal scaling). On the contrary, our proposed formulation allows for a more holistic representation of the VNF manager (VNFM) functionalities. The results show that our model has an attractive performance and the intelligent LCM mechanism ensures a stable and efficient system.

5.2 Future work

The current work extends the action space to a wider pattern, which includes a series of scaling operations as well as NFVI concerned LCM actions such as migration. However, we applied basic reinforcement learning algorithms to our model, which means we still have the potential for more complicated cases in the RL-based VNF-LCM problem. In future work, we will experiment with different RL tools to further enhance our scheme. We will extend our experiments with various LCM actions.

5.2.1 Deep reinforcement learning based VNF management in resource allocation model

To address the VNF management problem, the resource allocation model is a widely-discussed topic in the fields of NFV. The main challenge to deploy the VNF is to fulfill the requirement of a fast, scalable, and dynamic composition and allocations of VNFS to execute a network service. The resource allocation problem is the main challenge. As an extension of our work, a good start point is to apply DRL algorithms to our model. With the current problem model, if a DRL-based [81]algorithm can be applied, a more complicated VNF-LCM problem is more likely to be solved.

5.2.2 Multiple set of VNFs LCM problem

In our work, we simulate the proposed framework with several open source VNFs (container image). However, it needs more work to prove our framework fits well in other VNFs. A possible future research is to determine a full consideration of all kinds of VNFs and test their behaviors based on our VNF-LCM operations and adjust the reward function and algorithms.

5.2.3 VNF placement concerned VNF management

Another interesting topic is the VNF placement problem. Since we already discussed in the thesis that migration operation has its unique advantage over other VNF-LCM actions in some cases, it is natural for us to consider when and how to execute a migration operation for a VNF. More precisely, in practical cases, there must be cases with data center-based clouds and edge clouds. Suppose we would like to apply our mechanism. In that case, it will generate another optimizing problem, which is the VNF placement problem, where we have a concern not only on VNF efficiency and stability but also energy consumption concern and CAPEX/OPEX. With more VNF placement scenarios [82], it will be more practical for our work to gain a better performance.

5.2.4 A convinced VNF lifecycle management framework and NS orchestration

In our thesis, we solved the VNF-LCM problem via reinforcement learning by training RL agents to select the proper action to execute. However, there are external cases affecting the NS as well as the VNFs(i.e., NS crash, SLA changes). As a result, to manage the VNF framework well, it is necessary to promote a novel MANO framework which includes all the VNF management component as well as the NFVO layer and VIM layer components. This is also what researchers are focusing on in open-source projects such as OPNFV [83], ONAP [84].

References

[1] Network Functions Virtualisation (NFV), an Architectural Framework. . [Online]. Available: https://www.etsi.org/deliver/etsi$_g s/NFV/001_0 99/002/01.02.01_6 0/gs_N FV002v010201p.p$

[2] Network Functions Virtualisation (NFV), an Architectural Framework. . [Online]. Available: https://www.etsi.org/deliver/etsi$_g s/NFV - SOL/001_0 99/010/03.03.01_6 0/gs_N FV - SOL010v030301p.pdf$.

[3] S. Sultan, I. Ahmad, and T. Dimitriou. Container security: Issues, challenges, and the road ahead. *IEEE Access*, 7:52976–52996, 2019.

[4] *Reinforcement Learning: An Introduction*. 1998.

[5] Q. Li, X. He, M. Xu, Y. Jiang, and L. Wang. Unified middlebox model design and deployment with dynamic resources. *IEEE Transactions on Network and Service Management*, 15(3):1035–1048, 2018.

[6] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: Network processing as a cloud service. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, page 13–24, New York, NY, USA, 2012. Association for Computing Machinery.

[7] Network Functions Virtualisation (NFV), an Architectural Framework. . [Online]. Available: https://www.etsi.org/deliver/etsi$_g r/NFV - EVE/001_0 99/016/01.01.01_6 0/gr_N FV - EVE016v010101p.pdf$.

[8] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, 2015.

[9] A. OI, M. Nakajima, Y. Soejima, and M. Tahara. Reliable design method for service function chaining. In *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 1–4, 2019.

[10] H. Zhang, H. Ji, X. Li, K. Wang, and W. Wang. Energy efficient resource allocation over cloud-ran based heterogeneous network. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 483–486, 2015.

[11] M. Gerasimenko, D. Moltchanov, R. Florea, S. Andreev, Y. Koucheryavy, N. Himayat, S. Yeh, and S. Talwar. Cooperative radio resource management in heterogeneous cloud radio access networks. *IEEE Access*, 3:397–406, 2015.

[12] ETSI Network Function Virtualization. [Online]. Available:http://www.etsi.org/technologies-clusters/technologies/nfv.

[13] A. J. Gonzalez, G. Nencioni, A. Kamisiński, B. E. Helvik, and P. E. Heegaard. Dependability of the nfv orchestrator: State of the art and research challenges. *IEEE Communications Surveys Tutorials*, 20(4):3307–3329, 2018.

[14] B. Y. Lee and B. C. Lee. Analysis the architecture of vnfm (virtual network function manager). In *2015 17th International Conference on Advanced Communication Technology (ICACT)*, pages 336–340, 2015.

[15] P. L. Ventre, C. Pisa, S. Salsano, G. Siracusano, F. Schmidt, P. Lungaroni, and N. Blefari-Melazzi. Performance evaluation and tuning of virtual infrastructure managers for (micro) virtual network functions. In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 141–147, 2016.

[16] B. Lee and G. Lee. Service oriented architecture for sla management system. In *The 9th International Conference on Advanced Communication Technology*, volume 2, pages 1415–1418, 2007.

[17] Thomas Soenen, Felipe Vicens, José Bonnet, Carlos Parada, Evgenia Kapassa, Marios Touloupou, Eleni Fotopoulou, Anastasios Zafeiropoulos, Ana Pol, Stavros Kolometsos, et al. Sla-controlled proxy service through customisable mano supporting operator policies. In *IM*, pages 707–708, 2019.

[18] C. Parada, J. Bonnet, E. Fotopoulou, A. Zafeiropoulos, E. Kapassa, M. Touloupou, D. Kyriazis, R. Vilalta, R. Muñoz, R. Casellas, R. Martínez, and G. Xilouris. 5gtango: A beyond-mano service platform. In *2018 European Conference on Networks and Communications (EuCNC)*, pages 26–30, 2018.

[19] Network Functions Virtualisation (NFV), an Architectural Framework. . [Online]. Available: https://www.etsi.org/deliver/etsi$_g s/NFV - SOL/001_0$99/016/02.08.01$_6$0/$gs_N FV - SOL016v020801p.pdf$.

[20] A. Laghrissi and T. Taleb. A survey on the placement of virtual resources and virtual network functions. *IEEE Commun. Surveys Tuts.*, 21(2):1409–1434, 2019.

[21] K. Qu, W. Zhuang, X. Shen, X. Li, and J. Rao. Dynamic resource scaling for vnf over nonstationary traffic: A learning approach. *IEEE Transactions on Cognitive Communications and Networking*, pages 1–1, 2020.

[22] S. Lange, N. Van Tu, S. Jeong, D. Lee, H. Kim, J. Hong, J. Yoo, and J. W. Hong. A network intelligence architecture for efficient vnf lifecycle management. *IEEE Transactions on Network and Service Management*, pages 1–1, 2020.

[23] W. T. Scherer, S. Adams, and P. A. Beling. On the practical art of state definitions for markov decision process construction. *IEEE Access*, 6:21115–21128, 2018.

[24] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys Tutorials*, 18(1):236–262, 2016.

[25] 2007 institution of engineering and technology seminar on the pros and cons of using commercial 'off the shelf' components in aviation applications - cover page. In *2007 Institution of Engineering and Technology Seminar on The Pros and Cons of Using Commercial 'Off the Shelf' Components in Aviation Applications*, pages C1–C1, 2007.

[26] M. D. Ananth and R. Sharma. Cloud management using network function virtualization to reduce capex and opex. In *2016 8th International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 43–47, 2016.

[27] L. Mamushiane, A. A. Lysko, T. Mukute, J. Mwangama, and Z. D. Toit. Overview of 9 open-source resource orchestrating etsi mano compliant implementations: A brief survey. In *2019 IEEE 2nd Wireless Africa Conference (WAC)*, pages 1–7, 2019.

[28] O. Hahm, E. Baccelli, H. Petersen, and N. Tsiftes. Operating systems for low-end devices in the internet of things: A survey. *IEEE Internet of Things Journal*, 3(5):720–734, 2016.

[29] W. Wei and H. Yong. Application of decision support system in business enterprise management. In *2010 International Forum on Information Technology and Applications*, volume 3, pages 358–360, 2010.

[30] H. Khalili, A. Papageorgiou, S. Siddiqui, C. Colman-Meixner, G. Carrozzo, R. Nejabati, and D. Simeonidou. Network slicing-aware nfv orchestration for 5g service platforms. In *2019 European Conference on Networks and Communications (EuCNC)*, pages 25–30, 2019.

[31] S. Bijwe, F. Machida, S. Ishida, and S. Koizumi. End-to-end reliability assurance of service chain embedding for network function virtualization. In *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–4, 2017.

[32] B. Wang and M. Odini. Short paper: Lightweight vnf manager solution for virtual functions. In *2015 18th International Conference on Intelligence in Next Generation Networks*, pages 148–150, 2015.

[33] H. Yang, B. Mutichiro, and Y. Kim. Implementation of vnfc monitoring driver in the nfv architecture. In *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 683–685, 2017.

[34] R. Belter. Towards a service management system in virtualized infrastructures. In *2008 IEEE International Conference on Services Computing*, volume 2, pages 47–51, 2008.

[35] Bhagyalakshmi and D. Malhotra. A critical survey of virtual machine migration techniques in cloud computing. In *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*, pages 328–332, 2018.

[36] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz. Near optimal placement of virtual network functions. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 1346–1354, 2015.

[37] Franz Rothlauf. *Design of Modern Heuristics: Principles and Application*, volume 25. 01 2011.

[38] Ibrahim Osman and Gilbert Laporte. Metaheuristics: A bibliography. *Annals of Operational Research*, 63:513–628, 10 1996.

[39] X. Meng, V. Pappas, and L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *2010 Proceedings IEEE INFOCOM*, pages 1–9, 2010.

[40] Marcelo Caggiani Luizelli, Danny Raz, Yaniv ar, and Jose Yallouz. The actual cost of software switching for nfv chaining. 05 2017.

[41] Mathieu Bouet, Jeremie Leguay, and Vania Conan. Cost-based placement of virtualized deep packet inspection functions in sdn. pages 992–997, 11 2013.

[42] Tarik Taleb, Miloud Bagaa, and Adlen Ksentini. User mobility-aware virtual network function placement for virtual 5g network infrastructure. pages 3879–3884, 06 2015.

[43] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 500–507. IEEE, 2011.

[44] J. Upadhyaya and N. J. Ahuja. Quality of service in cloud computing in higher education: A critical survey and innovative model. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 137–140, 2017.

[45] Jia Rao, Yudi Wei, Jiayu Gong, and Cheng-Zhong Xu. Qos guarantees and service differentiation for dynamic cloud applications. *IEEE Trans. Netw. Service Manag.*, 10(1):43–55, 2012.

[46] Qi Zhang, Quanyan Zhu, Mohamed Faten Zhani, Raouf Boutaba, and Joseph L Hellerstein. Dynamic service placement in geographically distributed clouds. *IEEE Journal on Selected Areas in Communications*, 31(12):762–772, 2013.

[47] Milad Ghaznavi, Aimal Khan, Nashid Shahriar, Khalid Alsubhi, Reaz Ahmed, and Raouf Boutaba. Elastic virtual network function placement. In *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, pages 255–260. IEEE, 2015.

[48] Chunlin Li, Jianhang Tang, and Youlong Luo. Elastic edge cloud resource management based on horizontal and vertical scaling. *The Journal of Supercomputing*, pages 1–26, 2020.

[49] Windhya Rankothge, Helena Ramalhinho, and Jorge Lobo. On the scaling of virtualized network functions. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 125–133. IEEE, 2019.

[50] Hui Yu, Jiahai Yang, Carol Fung, Raouf Boutaba, and Yi Zhuang. Ensc: multi-resource hybrid scaling for elastic network service chain in clouds. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 34–41. IEEE, 2018.

[51] Hui Yu, Jiahai Yang, and Carol Fung. Fine-grained cloud resource provisioning for virtual network function. *IEEE Transactions on Network and Service Management*, 2020.

[52] V. Sciancalepore, F. Z. Yousaf, and X. Costa-Perez. z-torch: An automated nfv orchestration and monitoring solution. *IEEE Transactions on Network and Service Management*, 15(4):1292–1306, 2018.

[53] Hee-Gon Kim, Do-Young Lee, Se-Yeon Jeong, Heeyoul Choi, Jae-Hyung Yoo, and James Won-Ki Hong. Machine learning-based method for prediction of virtual network function resource demands. In *2019 IEEE conference on network softwarization (NetSoft)*, pages 405–413. IEEE, 2019.

[54] Rashid Mijumbi, Sidhant Hasija, Steven Davy, Alan Davy, Brendan Jennings, and Raouf Boutaba. Topology-aware prediction of virtual network function resource requirements. *IEEE Transactions on Network and Service Management*, 14(1):106–120, 2017.

[55] S. Jeong, H. Kim, J. Yoo, and J. W. Hong. Machine learning based link state aware service function chaining. In *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2019.

[56] J. S. Pujol Roig, D. M. Gutierrez-Estevez, and D. Gündüz. Management and orchestration of virtual network functions via deep reinforcement learning. *IEEE Journal on Selected Areas in Communications*, 38(2):304–317, 2020.

[57] S. Lange, N. Van Tu, S. Jeong, D. Lee, H. Kim, J. Hong, J. Yoo, and J. W. Hong. A network intelligence architecture for efficient vnf lifecycle management. *IEEE Transactions on Network and Service Management*, pages 1–1, 2020.

[58] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y. Liang, and D. I. Kim. Applications of deep reinforcement learning in communications and networking: A survey. *IEEE Communications Surveys Tutorials*, 21(4):3133–3174, 2019.

[59] Yi Tan and Guo-Ji Zhang. The application of machine learning algorithm in underwriting process. In *2005 International Conference on Machine Learning and Cybernetics*, volume 6, pages 3523–3527 Vol. 6, 2005.

[60] J. Yang, C. Zhou, Q. Chen, and T. Liu. An integrated design scheme of dynamic scheduling and control for networked control systems based on dynamic dwell time. In *Proceedings of the 33rd Chinese Control Conference*, pages 5774–5778, 2014.

[61] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR.

[62] M. Abu Alsheikh, D. T. Hoang, D. Niyato, H. Tan, and S. Lin. Markov decision processes with applications in wireless sensor networks: A survey. *IEEE Communications Surveys Tutorials*, 17(3):1239–1267, 2015.

[63] O. Sönmez and A. T. Cemgil. Sequential monte carlo samplers for model-based reinforcement learning. In *2013 21st Signal Processing and Communications Applications Conference (SIU)*, pages 1–4, 2013.

[64] Y. Chen, H. Yuan, and Y. Li. Object-oriented state abstraction in reinforcement learning for video games. In *2019 IEEE Conference on Games (CoG)*, pages 1–4, 2019.

[65] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[66] Hasselt. Double q-learning. in advances in neural information processing systems. pages 2613–2621, 2010.

[67] Z. Wang, X. Qiu, and T. Wang. A hybrid reinforcement learning algorithm for policy-based autonomic management. In *ICSSSM12*, pages 533–536, 2012.

[68] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[69] Y. P. Awate. Policy-gradient based actor-critic algorithms. In *2009 WRI Global Congress on Intelligent Systems*, volume 3, pages 505–509, 2009.

[70] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[71] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR.

[72] W. Wang, F. Ma, and J. Liu. Course tracking control for smart ships based on a deep deterministic policy gradient-based algorithm. In *2019 5th International Conference on Transportation Information and Safety (ICTIS)*, pages 1400–1404, 2019.

[73] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[74] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.

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

[76] Minghui Min, Liang Xiao, Ye Chen, Peng Cheng, Di Wu, and Weihua Zhuang. Learning-based computation offloading for iot devices with energy harvesting. *IEEE Transactions on Vehicular Technology*, 68(2):1930–1941, 2019.

[77] Xianfu Chen, Honggang Zhang, Celimuge Wu, Shiwen Mao, Yusheng Ji, and Medhi Bennis. Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. *IEEE Internet of Things Journal*, 6(3):4005–4018, 2018.

[78] Deval Bhamare, Andreas Kassler, Jonathan Vestin, Mohammad Ali Khoshkholghi, and Javid Taheri. Intopt: In-band network telemetry optimization for nfv service chain monitoring. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2019.

[79] Warwick Masson, Pravesh Ranchod, and George Konidaris. Reinforcement learning with parameterized actions. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 1934–1940. AAAI Press, 2016.

[80] Nan Ding and Radu Soricut. Cold-start reinforcement learning with softmax policy gradient. In *Advances in Neural Information Processing Systems*, pages 2817–2826, 2017.

[81] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau. 2018.

[82] D. Amaya, Y. Sumi, S. Homma, T. Okugawa, and T. Tachibana. Vnf placement with optimization problem based on data throughput for service chaining. In *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, pages 1–3, 2018.

[83] A. Boubendir, E. Bertin, and N. Simoni. On-demand dynamic network service deployment over naas architecture. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 1023–1024, 2016.

[84] V. Q. Rodriguez, F. Guillemin, and A. Boubendir. Network slice management on top of onap. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–2, 2020.