



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

Thesis for the Degree of Masters

# Efficient Autoscaling of VNF(s) in NFV environment

**Asif Mehmood**

Department of Computer Engineering

GRADUATE SCHOOL

JEJU NATIONAL UNIVERSITY

June 2019

# Efficient Autoscaling of VNF(s) in NFV environment

Asif Mehmood

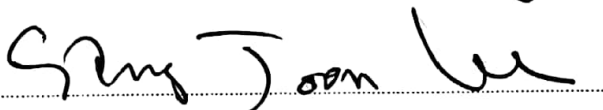
(Supervised by Professor Sang-Joon Lee)

Submitted to the Department of Computer Engineering and the Faculty of Graduate School of Jeju National University in partial fulfillment of the requirements for the degree of Master of Computer Engineering

2019.06

This thesis has been examined and approved.

  
Wang-Cheol Song, Professor, Jeju National University

  
Thesis Supervisor, Lee, Sang-Joon, Professor, Jeju National University

  
Thesis Director, Byun, Yung-Cheol, Professor, Jeju National University

Department of Computer Engineering

GRADUATE SCHOOL

JEJU NATIONAL UNIVERSITY

Dedicated to  
*My Parents and Teachers*

## Acknowledgements

This study is wholeheartedly dedicated to my beloved family, who have been a source of inspiration and by continually providing their spiritual, emotional, and financial support.

I would also like to thank the most respected and kind supervisor Wang-Cheol Song for his support, guidance and advice during the whole project. This thesis without the precious advices of my supervisor wouldn't have been completed, that were provided time to time on regular basis. I would like to acknowledge that he played a vital role in building up my ability to perceive knowledge in multiple perspectives.

I feel indebted to many of the colleagues and the friends at Jeju National University for providing a very professional and friendly environment. I can not mention all of the fellows but there are a few names I would like to mention as their role was very special. I am grateful to Dr. Afaq Muhammad, Dr. Israr Ullah, Dr. Muhammad Fayaz, Faisal Mehmood, Shabir Ahmad, Javier Diaz, Talha Ahmed.

Last, but not the least, a very special role that this country played in providing me an exposure to study abroad and to fulfil my dreams is not forgettable. All the people living in this country, whose names I couldn't mention are very special to me I want to thank all of you to play a very useful role in my life.

## Abbreviations

API(s)	Application Program Interfaces
ATM	Automated Teller Machine
BAAS	Bit-rate Aware Auto Scaling
BP	Business Process
BSS	Business Support System
CAPEX	Capital Expenditure
CiaB	CORD-in-a-Box
CORD	Central Office Rearchitected Datacenter
CORD-VTN	CORD Virtual Tenant Network
CPS	Clocks Per Cycle
CPU	Central Processing Unit
EMS	Element Management Service(s)
EPC	Evolved Packet Core
ETSI	European Telecommunications Standards Institute
GUI	Graphical User Interface
IaaS	Infrastructure as a Service
IBN	Intent Based Networking
IMS	IP Multimedia Subsystem
JSON	JavaScript Object Notation
KVM	Kernel-based Virtual Machine
LIBVIRT	library for virtualization
LTE	Long Term Evolved
LXD	Linux [Container] Docker
M-CORD	Mobile Central Office Rearchitected Datacenter
MaaS	Monitoring as a Service
MaaS-target	Metal as a Service targets
MANO	Management and Orchestration
MCN	Monitoring and Control Network
MySQL	My Structured Query Language
NIC	Network Interface Card
NFD	Network Function Descriptor
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure

NFVO	Network Function Virtualization Orchestrator
NS	Network Service
NSD	Network Service Descriptor
NSSF	Network Slice Selection Function
OAI	Open Air Interface
OAISIM	Open Air Interface Simulator
ONF	Open Networking Foundation
ONOS	Open Networking Operating System
OPEX	Operational Expenditure
OPNFV	Open Platform for NFV
OSS	Operations Support System
OVF	Open Virtualization Format
PMU	Performance Monitoring Unit
RAM	Random Access Memory
REST	Representational State Transfer
SDN	Software Defined Networking
SDNC	SDN Controller
TOSCA	Topology and Orchestration Specification for Cloud Application
vEPC	Virtual Evolved Packet Core
vHSS	Virtual Home Subscriber Server
VIM	Virtual Infrastructure Manager
vMME	Virtual Mobility Management Entity
VM	Virtual Machine
VND	Virtual Network Deployment
VNF	Virtual Network Function
VNFC	Virtual Network Function Component
VNFM	Virtual Network Function Manager
VoIP	Voice over IP
vSPGW-C	Virtual Software Packet Gate Way Control Plane
vSPGW-U	Virtual Software Packet Gate Way User Plane
QoS	Quality of Service
XOS	Everything as a service Operating System
5G	5th Generation

## Table of Contents

#	Sections	Page
-	Acknowledgements	iv
-	Abbreviations	v
-	Table of Contents	vii
-	List of Figures	ix
-	List of Tables	x
-	Abstract	1
1	<b>Introduction</b>	<b>2</b>
2	<b>Literature Review</b>	<b>5</b>
2.1	- General Literature	5
2.1.1	- - Cloud	5
2.1.2	- - SDN and NFV	7
2.1.2.1	- - - SDN - Software Defined Networking	7
2.1.2.2	- - - NFV - Network Function Virtualization	8
2.1.3	- - MANO Architecture	11
2.1.4	- - M-CORD	12
2.2	- Related work Literature	13
3	<b>Monitoring and Autoscaling System</b>	<b>18</b>
3.1	- Modifications proposed	20
3.2	- Layers	21
3.2.1	- - Infrastructure Layer	21
3.2.2	- - Management Layer	22
3.2.3	- - Application Layer	23
3.3	- Modules	24
3.3.1	- - Intent Based Autoscale Application	27
3.3.1.1	- - - Information Assembler	27
3.3.1.2	- - - Autoscale Controller	29
3.3.1.3	- - - Configuration Invoker	32
3.3.2	- - Monitoring Microservice	33
3.4	- Configuration of System	34
3.5	- Specifications	35
3.5.1	- - Application specifications	35
3.5.2	- - Microservice specifications	35



<b>4</b>	<b>Evaluation and Results</b>	<b>36</b>
4.1	- CORD Configuration Steps	37
4.1.1	- - Preparation-targets	37
4.1.2	- - MaaS-targets	37
4.1.3	- - XOS-targets	38
4.1.4	- - ONOS-targets	38
4.1.5	- - OpenStack-targets	38
4.1.6	- - Post Onboarding-targets	38
4.1.7	- - Additional CiaB-targets	38
4.2	- Evaluation Metrics	39
4.2.1	- - Assigned CPU Usage	39
4.2.2	- - Overall CPU Usage	39
4.3	- Evaluation Results	40
4.3.1	- - Assigned CPU Usage	40
4.3.1.1	- - - Average stats for each VNF	40
4.3.1.2	- - - Average stats for all VNFs	44
4.3.2	- - Overall CPU Usage	48
4.3.2.1	- - - Average stats for overall CPU	48
<b>5</b>	<b>Conclusion and Future Work</b>	<b>51</b>
<b>6</b>	<b>References</b>	<b>52</b>

## List of Figures

#	Name	Pp.
Figure 1:	OpenStack Overview - Infrastructure as a Service	6
Figure 2:	SDN Architecture - Abstracted Logical View of SDN	7
Figure 3:	NFV Architecture - An Overview of NFV Elements	9
Figure 4:	NFVO interaction with OSS/BSS and VNFM	10
Figure 5:	NFV MANO - Management and Orchestration	11
Figure 6:	Overall Architecture for Autoscaling VNFs	19
Figure 7:	Monitoring microservice comm. with sync. via an API	22
Figure 8:	Steps - Overall mechanism between the sys. components	24
Figure 9:	Information Assembler Database Tables	27
Figure 10:	Autoscale-Controller - Flow chart of decision-making prc.	29
Figure 11:	Autoscale-Controller - Mechanism of decision-making prc.	30
Figure 12:	Autoscale-Controller - Algorithm of decision-making prc.	31
Figure 13:	Configuration Invoker comm. with Autoscale Controller	32
Figure 14:	Configuration and Requirements of the System	34
Figure 15:	CORD installation steps	37
Figure 16:	Usage of Assigned CPU/VNF for first cycle	41
Figure 17:	Usage of Assigned CPU/VNF for second cycle	42
Figure 18:	Usage of Assigned CPU/VNF for third cycle	43
Figure 19:	Usage of Assigned CPU/VNFs for first cycle	45
Figure 20:	Usage of Assigned CPU/VNFs for second cycle	46
Figure 21:	Usage of Assigned CPU/VNFs for third cycle	47
Figure 22:	Usage of Total CPU (bar)	49
Figure 23:	Usage of Total CPU (line)	49
Figure 24:	Core allocation by applying different workloads on VM	50

## List of Tables

#	Name	Pp.
Table 1:	Specifications of Physical and Deployment system	36
Table 2:	Cycle 1: Usage percentage for each VNF at time interval 't'	40
Table 3:	Cycle 2: Usage percentage for each VNF at time interval 't'	42
Table 4:	Cycle 3: Usage percentage for each VNF at time interval 't'	43
Table 5:	Cycle 1: Usage percentage for all VNFs at time interval 't'	44
Table 6:	Cycle 2: Usage percentage for all VNFs at time interval 't'	46
Table 7:	Cycle 3: Usage percentage for all the VNF at time interval 't'	47
Table 8:	Average usage percentage of Total CPU at time-interval 't'	48

## Abstract

Autoscaling is one of the principal objectives for the intent based future networks. Scaling the cloud resources in different situations was dependant on a cloud expert administrator. This dependancy led to the demand of automating the process of scaling resources. Specifically, one of the key players that drove and came up with the idea of autoscaling network resources was SDN, Invention of SDN led to the possibility of autoscaling in cloud environments because of its capability to separate the control plane from the data plane. Automating the process of scaling network resources was not possible without the need of an administrator but as the networks evolved with the invention of SDN, it led to provide connectivity between the virtualized network resources. To use the network resources efficiently is the focal interests of future networks. The proposed system comprises of an intent based autoscaling application, monitoring microservice and a few modifications proposed to the elementary management services. The application takes the network resource's key factors into account such as cores in execution-times, CPS, RAM, Hard disk, hyper-threads etc, and keeps weight factor into consideration for virtualized/physical core, while deciding the resource scalability after the intervals. All the factors mentioned above, ensure to estimate the assignation of resources up to optimal. This mechanism provides us a platform with efficient computation capabilities. While the necessary monitoring keeps track of the information from all over the virtual network resources assigned to a specific tenant. The proposed monitoring service being at the management layer provides us a way to fetch realtime data from the resources in a less latent approach. For the overall system, M-CORD (a Framework developed by ONF) was used to deploy network functions inside compute machines and integrated the proposed system with it to evaluate the aforementioned key factor such as efficiency, in terms of CPU, RAM usages for specific intervals.

# 1 Introduction

We present the introduction by starting with the problem-statement and then jump towards explaining our motivation to introduce the system objectives. The way the content is organized, is also described at the end of this section.

As central offices being put on the edge clouds makes them of more vital importance when it comes to the performance of the network services. Mentioning the telco's infrastructure, it is not wrong to say that it requires a lot of expense in the infrastructure to provide the network resources. Making the orchestration [9] process automatic [2] and efficient provides us the motivation to introduce a solution of autoscaling. One of the use case for automation is autoscaling, which means to automatically scale in/out the network resources in the time when it is needed.

Scaling services in cloud [10] environments is of importance no doubt, but the quality of service that the network services should provide to the vendors/clients must be guaranteed and is of vital importance. As it all depends upon the usage of resources responsible to serve the traffic coming from end users, so ignoring the key factors including the computing resources whether they are virtual/physical can either result in over utilization or under utilization of the resources. Another factor of interest is the weight factor [11] used to calculate the accurate resources to be assigned to a tenant/vendor through the use of an efficient algorithm proposed in terms of usage the amount of resources over the whole interval of time as well as the efficiency into account.

So, the architecture in itself proposes an efficient way to place the modules in such a way that it provides a less latent model of request handling between the microservices. This model is defined through well defined API(s) [12] which provides us a way to program such services which are platform/language independent through the use of JSON [13] formatted requests and responses.

The primary motivation and objective of this research is to make the process of autoscaling efficient. Efficiency of the system comes through the proper allocation of computing resources including CPU, RAM, Hard disk in the telco networks, which run network services on them. System comprises of an Intent [14] Based Autoscaling application and Monitoring [4] microservice. Firstly, the application comprises of a database store that is synchronized with the resource usage of each of the network function instances. The usage information of resources is provided through a proper and well defined interface between the Monitoring [4] microservice and the Intent [14] Based Autoscaling application.

The realtime data and the policies defined by the contract [14], both define when to scale in/out which of network instances. This improves the usage of resources by an intelligent decision-making algorithm. i.e. efficiency of the system. The system also takes the factor into account that the user defined contract is not violated.

By following the above mentioned objectives and procedures, it is intended to provide a mechanism of autoscaling which is efficient. Secondly, through well defined procedures in the architecture, the plan is to show efficiency by comparing the assigned resources with the used resources. The evaluation of the system performance is proven to be better. To show the future enhancements that can be made to this system are also one of the objectives of this document.

The content of this thesis is organized from Chapter 2 - Chapter 6 as:

Chapter 2 - This section starts by providing the necessary information of cloud, specifically the OpenStack [15] which provides IaaS. After enlightening the cloud infrastructure, a detailed literature for the SDN [3] and NFV [9,16] is provided. M-CORD [6,7] platform is enlightened in details with its technical architecture.

Chapter 3 - This section covers the details of proposed design for this document, with the interfaces it has and then the minor details of the mechanism is described. The layered architecture and the placement of applications is also discussed in terms of benefits.

Chapter 4 - This section covers the steps taken to evaluate the results. Firstly, the setup environment is discussed. The necessary steps/targets required to configure the test environment is also explained in detail. At the end, the metrics used to evaluate are shown and their impacts on the system are analysed with discussion.

Chapter 5 - Finally, concluded the whole thesis's purpose and provided the necessary direction for future to be followed.

Chapter 6 - References section provides all the documents, papers, open source projects that were used as help in different ways such as for a referral, details and implementations.

## 2 Literature Review

We have divided the literature into 2 sections as there are some vast topics to be discussed earlier rather than to start jumping towards the related work. For this purpose, we present them as followed:

### 2.1 General Literature

The general literature introduces each and every detail of the concepts used in this document. We also tried to explain them in a way to make our project more easily understandable.

#### 2.1.1 Cloud

As the clouds [10] play a very important and vital role to provide either software, platform or infrastructure as a service. The main theme of cloud is to provide a flexible environment to the service providers, so that they don't have to worry or think about the technical problems that could arise while upgrading or migrating the system. It is clear from the figure below which allows the cloud resource provider to partition the infrastructure into multiple isolated environments. This further gives the flexibility to achieve platform independency. This attribute of cloud improves the efficient resource usage of the infrastructure.

The below figure shows an example of Infrastructure as a Service i.e. OpenStack. The way this platform works is that it has services deployed across the compute nodes as distribute services. It has some of the services deployed on the controller node. A few names of the services that are used in this project are Keystone, Glance, Nova, Neutron, Horizon etc [15].

Keystone authenticates the requests made among the services of OpenStack cloud controller. Glance is another important service that OpenStack has, which stores the images of the virtual machines. Nova provisions the VMs by fetching Glance images on a specific network. Neutron is there only to create network(s),



subnet(s) and is responsible to handle the requests related to network management. By default it has some of the basic capabilities of networking embedded in its own project, but it can be integrated with SDN to enhance the functionality of networking in cloud environment. Horizon service in order to make the IaaS to work properly provides a GUI.

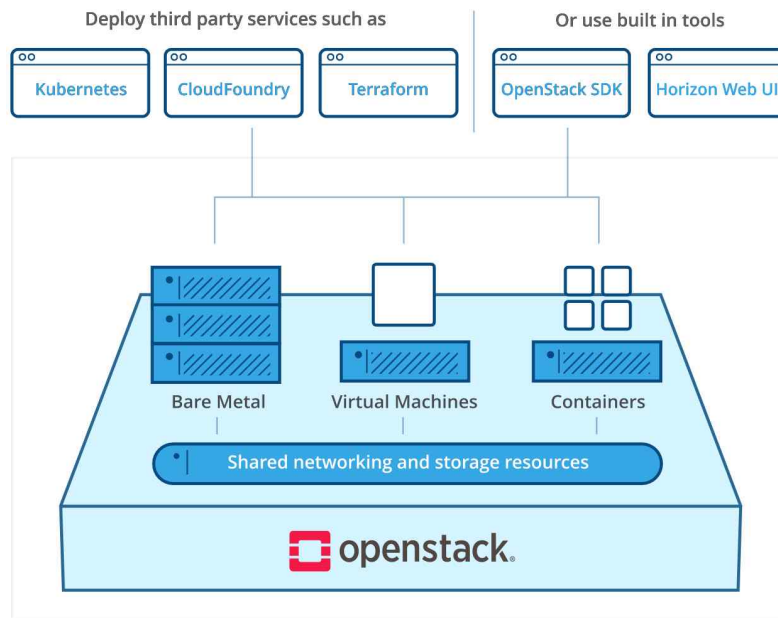


Figure 1 OpenStack Overview - Infrastructure as a Service

Concluding all of the services that it provides, this project (OpenStack [15]) at the end tends to provide infrastructure as a service. On this infrastructure, it provides the ability to create virtual resources such as virtual-machines, virtual-networking and virtual-storage which are logically isolated from the other virtual-resources deployed on the same infrastructure. This architecture of providing infrastructure as a Service retains flexibility both for the clients as well as the vendors. The vendors flexibility means that they can add their own custom services as on top of OpenStack to manage their infrastructure according to their own custom requirements.

### 2.1.2 SDN & NFV

First we give a brief overview of what SDN is and how it relates to our work and in the second subsection we put more details on the NFV architectures available and how are these related to our work. How much the current platform differs from NFV is also discussed here.

#### 2.1.2.1 SDN - Software Defined Networking

Software Defined Networking had its deep root concepts from the very early age of networks. The need of this kind of architecture was due to the high cost of hardware and even after investing too much money on the hardware, the network operators were not able to bring innovativeness without the need of a new hardware. The SDN Architecture had its concept in ATM(s) [23] and in some of the traditional systems but approaches were not officially standardized.

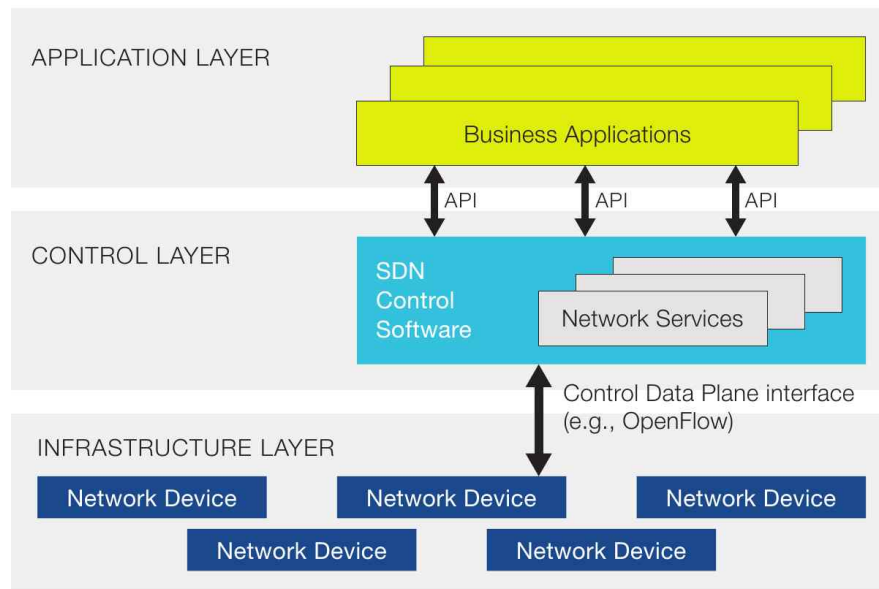


Figure 2 SDN Architecture - Abstracted Logical View of SDN

First starting with the network devices that build up the network infrastructure layer. These devices could be switches, routers, radio-access points or any kind of device that is needed to build up a complete network to provide connectivity. These devices in the case of SDN, only have the role to forward the data-plane

traffic rather than to take the decisions autonomously like traditional network devices. This gives the control to some other party [3] that we are going to talk about in the next paragraph. Second layer of the SDN is also known as the brain of SDN. The intelligence that was separated from the infrastructure is now a part of SDN Controller. This is the part where Network Services exist, and are in the form of a software. This separation of control-plane from the data-plane allows us to control the data-plane by a variety of network applications. Then comes the last and top most layer of SDN Architecture. It is where the northbound applications run and communicate with the controller via REST API interfaces. This gives the flexibility to developers to use any type of language to use for developing the application. OpenStack or VMware vCloud Director could be integrated via the REST API(s) exposed on the northbound. Then these kind of applications could be useful to manage the network services in a cloud. Other example of network applications could be given such as a custom routing application and monitoring application.

#### **2.1.2.2 NFV - Network Function Virtualization**

NFV is often confused with the term SDN. So in this part of the literature, first we're going to have a look at what NFV [16,17] is. Then we will observe the differences between the SDN and NFV. The main goal of NFV is to provide a cheaper platform to control the virtual networks using both the attributes of a Orchestration techniques, Cloud and SDN together. A lot of standards have been defined from different organizations of which the most prominent organizations are ETSI [17] and OPNFV [18]. Both are explained with details in the later sections. Now we put some light on the key elements involved in the NFV architecture. The below shown figure has 3 layers. Starting from bottom most layer, it can be seen that it comprises of NFVI and VIM [17]. NFVI is known Network Function Virtualization Infrastructure while VIM is called a Virtual Infrastructure Manager. VIM is a manager for the cloud infrastructure manager. The most popular example of cloud IaaS is OpenStack [15]. OpenStack provides a way to create isolated/virtual environments (for storages, networks,

VMs) on the infrastructure via the services it provides. NFVI has a virtualization layer in between the virtual resources and the actual resources. Actually this virtualization layer is called as a hypervisor/container-engine in the case of a VM/container respectively. SDN switch works as network hypervisor which enables the separation of control plane and data plane. Virtual storage hypervisor [25] runs as an agent inside the VM and maps it onto the physical infrastructure.

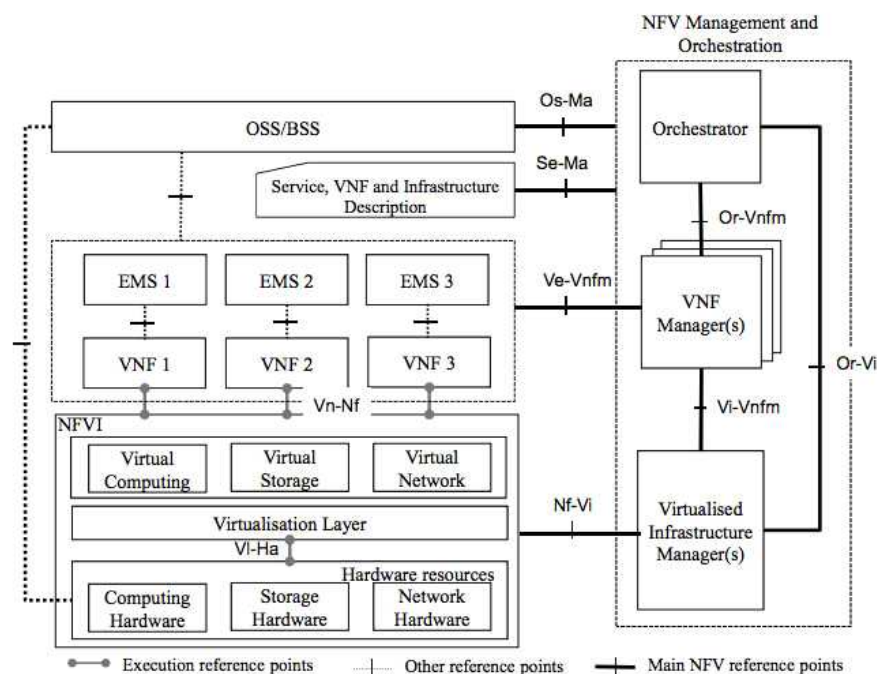


Figure 3 NFV Architecture - An Overview of NFV Elements

The second layer is the virtual resources. The first component in this layer is VNFM (Virtual Network Function Manager). It is responsible to manage the life-cycle of the VNF(s) that are specific to a VNF Manager. Each VNF is associated to a specific VNFM. There can be multiple VNFM(s) under the orchestrator to handle multiple domains. It takes care of the instantiation of the VNF(s), scaling them up or down on the virtual layer. It can also take necessary actions such as updating, upgrading or the termination of the VNF(s). VNFM also has other interfaces for interacting with EMS(s) and the Orchestrator. The

main difference between the VNFM and EMS is that the VNFM manages the VNF such as scaling them up or down while the EMS manages the functional part of the VNF. An EMS has usually a one to one relation with a VNF but it can span up to manage multiple VNF(s). These VNF(s) are deployed on the NFVI.

The third layer in NFV comprises of OSS/BSS and Orchestrator. OSS/BSS just lets the network organization employee(s) to interact with the whole system. These tasks or responsibilities could be to apply policies to the underlying system or it could allow the businesses to define the network services via the help of an orchestrator [9]. Orchestrator is the one which holds the network service definitions that are described as NSD(s) and NFD(s) [26] from within different domains of network operators. These NSD(s) and NFS(s) are translated by the Orchestrator (via VNFM, VIM and NFVI) into applicable configurations to the underlying platforms. The NFV Orchestrator abbreviated as NFVO [19] has information of NS-catalogue, VNF-catalogue, NFV Instances and NFVI resources. Network service and VNF catalogues contain all information of the network services and VNF(s). The two main responsibilities of NFVO are resource and network service orchestration.

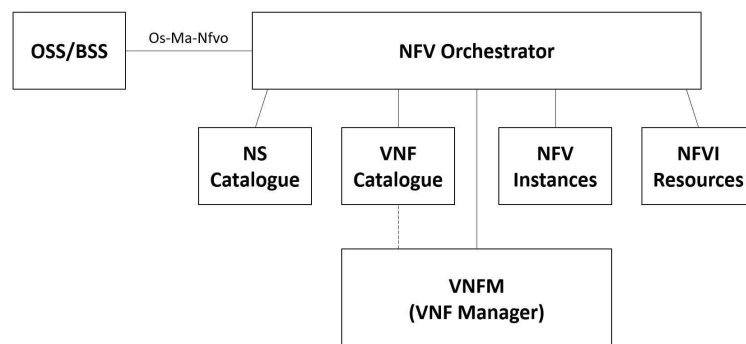


Figure 4 NFVO interaction with OSS/BSS and VNFM

### 2.1.3 MANO Architecture

In this section we describe the MANO [19] as a subpart of NFV Architecture. It comprises of an Orchestrator [9], VNFM and a VIM. The orchestrator has the responsibility to orchestrate two things. One is resource and the second is the network services. It interacts with VIM in order to orchestrate the necessary virtual resources such as compute (VMs), network resources.

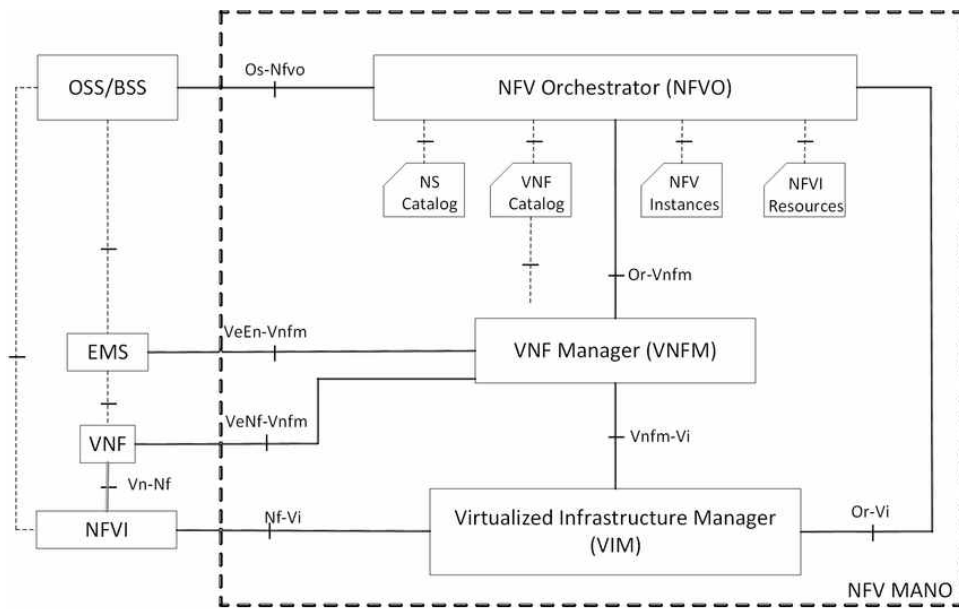


Figure 5 NFV MANO - Management and Orchestration

Second is the network service to be orchestrated. This is achieved by the interaction of an orchestrator with the VNFM. For this purpose, the orchestrator requires NS-catalogue and VNF-catalogue.

Then VNFM communicates with the VIM to look for the resources if they have been setup for deployment or not. Until they are setup, VNFM waits and after the procedure is completed, it deploys the Element Managers inside the VNF or the VM. These element-managers are responsible to manage the functions inside a VNF. These element-manager deploy the function initially and also manage the functions while they are running too.

#### 2.1.4 M-CORD

This section is to familiarize you with the M-CORD [6] terminologies. The owner organization of this project is ONF. It's goal is to provide a framework which can orchestrate everything as a network services. M-CORD is the framework on which any vendors network services can be deployed. We as a student chose an opensource and free solution i.e. OAI [28,30] Network Services. We used their eNB and vEPC network services for deploying the access and core network functionality and our own developed NSSF network service.

This framework with the help of their own developed XOS [20] enables the stakeholder to run every thing as a service. Even the most common opensource projects such as OpenStack and ONOS run under XOS as a services, thus allowing to integrate multiple projects to communicate with each other. This ability of running each thing on the same level allows flexibility in terms of your service exposure and openness as well. First we'll start by explaining how OpenStack and ONOS runs as a service under the roof of XOS. For deployment, consider 3 nodes corddev, head and compute nodes. The head-node contains the synchronizers services. In case of OpenStack [15], ONOS and VNFs, it has synchronizers for each of them. The purpose of OpenStack in this framework is that it handles the management of VMs. ONOS [21] has two of the applications highly customized e.g vRouter [31,32,33] (which is responsible to route the traffic from one subnet to other subnet) and CORD-VTN [31,32,34] (which is responsible to manage the isolated networks).

XOS is the manager of all the services. It is the brain of this architecture. The synchronizers are responsible to perform the steps to setup the state of an entity to the desired state with some predefined steps. Synchronizers does the job of an Elementary management services to ensure the management of system.

## 2.2 Related work Literature

In this section, we discuss about the related work that has been done is currently being researched on.

The author in the paper [1] uses the Tacker to showcase the benefits of NFV Architecture. Throughout this paper, autoscaling functionality and the reasons of deploying VNFs in virtual-machines has been discussed. It also shows how we can reduce time of automatic-provisioning or autoscaling resources dynamically.

Author of paper [2] describes a mechanism to autoscale the resources. Specially, the data plane functions such as SGW, PGW and how they can affect the CAPEX and OPEX. The unique thing about this document is that they propose a BAAS. This autoscaling mechanism maintains an accurate and precise user-equipment bit-rate required in the network-slices without over-provisioning of the resources.

Monitoring in NFV environment is much of an importance as the response-time of monitoring could dynamically change due to the nature of cloud. In this paper [4], they propose a solution of VNFC monitoring driver. Instead of their approach, we used API based interfaces to achieve platform independency for the network developers. This lets the developers and organizations to deploy their services (monitoring-services) on platforms irrespective of platforms.

In the paper [5], we proposed a solution in NFV environment that comprises of an application to autoscale as well as a monitoring solution, as our contribution. The importance of orchestration has been mentioned for the future networks. The monitoring importance is related with the orchestration mechanism, as it is of vital importance to provide real-time data correctly and without any delay.



In the thesis [6], the author proposes a layered solution termed as VIM Adaptation Layer for CORD. The author describes the need for a generic layer in order to define a generic XOS models. This will also allow to us to define generic synchronizers which will merge the functionality of multiple Virtual Infrastructure Managers and Multiple SDNC. The author also describes the way of installing M-CORD platform which we also discuss in our section relevant to the M-CORD platform.

In the mentioned paper [8], we as contributor explained a way to manage network slicing inside the M-CORD framework. We also developed a prototype to demonstrate the LTE Advanced architecture. We used the OpenStack and ONOS based integrated test-bed. The slicing in this prototype was done at transport level. This paper helps us to understand transport network slice management.

In the paper [9], author highlights the need of dynamic adaptation of SFC paths due to the high availability requirements. They propose an SDN orchestrator that adapts the service chaining paths to reduce congestion in the network. Along with this mechanism, it ensure the QoS provided.

Writer of the paper [10] describes the issues and techniques that are used to fulfil the demands of network users at peak workloads. These needs require a central/autonomic management system which the author proposes. They provide a mechanism of provisioning virtual machine resources dynamically at the peak workload time. They provided a solution that guarantees the optimal power consumption in the time of high/low traffic both.

In the paper [11], author proposes an algorithm in which autoscaling decision is made by trading off between the cost an performance. For this purpose, they use the weight factors, capacity of VNFs, threshold, job arrival rate, service rate for each server, setup rate (time required) for each VM.

Author highlights two issues in the current systems in this paper [13]. Consistency and data-integrity are achieved via subsystems that exchange information in JSON format. This approach also lets two different systems to be interoperable via a standardized message exchange format i.e. JSON.

In the paper [14], we proposed and developed a solution in which we give the facility of defining high-level contractual information to the subscriber as well as the network operator. This approach fulfills the requirement of user/operator driven system for the provisioning of a network slice. We developed an IBN application for this purpose which was responsible to detect and resolve conflicts via the use of our vocabulary-store that stored the necessary information of the supported network-services and architectures.

In the book [15], author defines the necessary literature related to the NFV (Network Function Virtualization) required for the beginner, intermediate level learners. Author also describes the need of an SDN inside the defined NFV architecture with different perspectives of a user.

The paper [20] proposing an application defined operating-system solution for the Datacenter computing platforms. As from the name XOS, it refers to an Operating System which runs everything as a service. This provides good scalability and strong performance with isolation. This kind of mechanism in order to orchestrate everything on the same level achieves high level of integrity among the services. It is a Service Orchestrator in M-CORD framework that provides a GUI to interact with as well it exposes REST API interfaces in order to allow the integration of outside systems too. We used these XOS provided REST API(s) to integrate our IBN application [14] with M-CORD.

The white paper [24] discussing about new norms for the networks has a section about the Openflow agent. This document also puts some of the light on the protocol that is used for the communication between switches and the Software Defined Controller. The benefits of having Openflow as a southbound protocol are discussed in this paper such as central control for multiple vendors, less complexity in networking, higher rate of innovation, increased security, more granular network control and better user experiences.

In the paper [26], author proposes a way of describing a virtual network deployment termed as VND. It proposes an extended OVF solution that is constraint based. It discusses the behaviour of elements involved in the virtual networks which are nodes, links, networks and then put two viewpoints: requested service quality and quality of service offered.

From this paper [35], we refer to show or raise the point of how much dependent memory and core requirements are on the applications hosted on it. So the first thing that comes into mind is that the application has a number of functions to perform so why not take the execution-time of the specific actions for calculating the resources requirements. The author talks about the parallelism involved in the applications which could be either inter or intra application. These two aspects are important for estimating the core for a virtual machine as there are other applications running on the virtual-machine as well as the same application might have some processes which are running concurrently for achieving better performance.

Firstly, the author discusses the dependencies and communication between tasks within a BP, Secondly it takes into account several objectives like to minimize the execution time/cost of a program and maximizing the resource utilization. Other factors when involved such as unavailability of resources and due to the overloaded networks which make the scheduling algorithm more complex. The solution in this paper [36] is the proposal of an adaptive BP in the cloud.

The author in this paper [37] proposes an extensible orchestration technique, specifically for the IP multimedia subsystem i.e. IMS. IP multimedia services which could be VoIP and Calling etc. The author used MCN Monitoring as a Service MaaS and OpenBaton for building up a complete setup for monitoring and orchestration together respectively. This is completely compliant to ETSI NFV's MANO specifications

Although the current NFV implementations done by the current organizational bodies are improving day by day but the service lifecycle is not managed well enough which includes the responsibility of reducing time of resource creation, repairing and deletion. Author in the paper [38] proposes a solution fulfilling the aforementioned attributes which could overall improve the lifecycle management for the VNFs in NFV Architecture.

The author of this paper [39] proposes a new solution that is a two time-targeted scheduling scheme. This involves both systems which are to be deployed sequentially as well as in a distributed deployment. At the end they compare their system's performance, in the form of an number which is an optimized number to show how much resources are to be assigned.

In the paper [40] discusses the impacts of hyperthreading overhead on the processing capabilities. They calculated the efficiency by providing a comparison between the hyper-threaded and single-threaded processors with the use of PMU data.

### 3 Monitoring and Autoscaling System

We propose a solution to autoscale [35] the VNF(s) inside M-CORD framework with the help of our custom monitoring application. The VNF(s) are under/over utilized in some cases causing unfairness on the cloud data-centers for other operators. We intend to provide a solution for the following problem to distribute the resources among different tenants depending on the policies as well as the resource usage by the VNF(s). The resource usage parameters are the uniqueness of this system that the other systems don't take into account. The design of the system is followed by this section. The conceptual layers are explained in the coming subsections.

The design presented below defines the architecture and overview of how modules are logically separated on different layers of the design. First comes the Application layer which is at the top of architecture and it includes our Autoscaling Application having 3 sub-modules named as Configuration-invoker, Autoscale-controller and Information Assembler. Then comes the Management layer where our custom monitoring application lies which reduces the latency of processing in the current mechanisms. The elementary management services [17] often termed as synchronizers in the world of ONF, have been modified to adopt the situation. Each synchronizer exposes an API interface which provides a way to fetch the information of resources used. This information includes the number of requests received and the memory related parameters. Using this information, we estimate the resources to be assigned for the next cycle e.g. CPU, RAM etc. We use the weight factor [10,11] to assign the cores as it is obvious that the virtual core's performance is not equal to the physical core's performance, specifically in the worst cases. As the mobile networks is a critical area where we can't take risks but also to utilize the resources up to optimum. We provide a mechanism and an algorithm to solve the problems addressed in an efficient manner. This approach as a prerequisite leads us to calculate the number of CPU(s) to be assigned upon the bases of the execution-times [36] for

the programs that run inside the VNF(s). This approach as a result frees up the resources that not need to be assigned, letting the other tenants or network operators to use the resources. Another that results using our proposed architecture is that the resources that are assigned are used up to the optimum level. This study and demonstration proves itself in the Experiments and Evaluation's section to be worthy. Another aspect of our application is that you can customize it in a really simple way which offers flexibility to the developers.

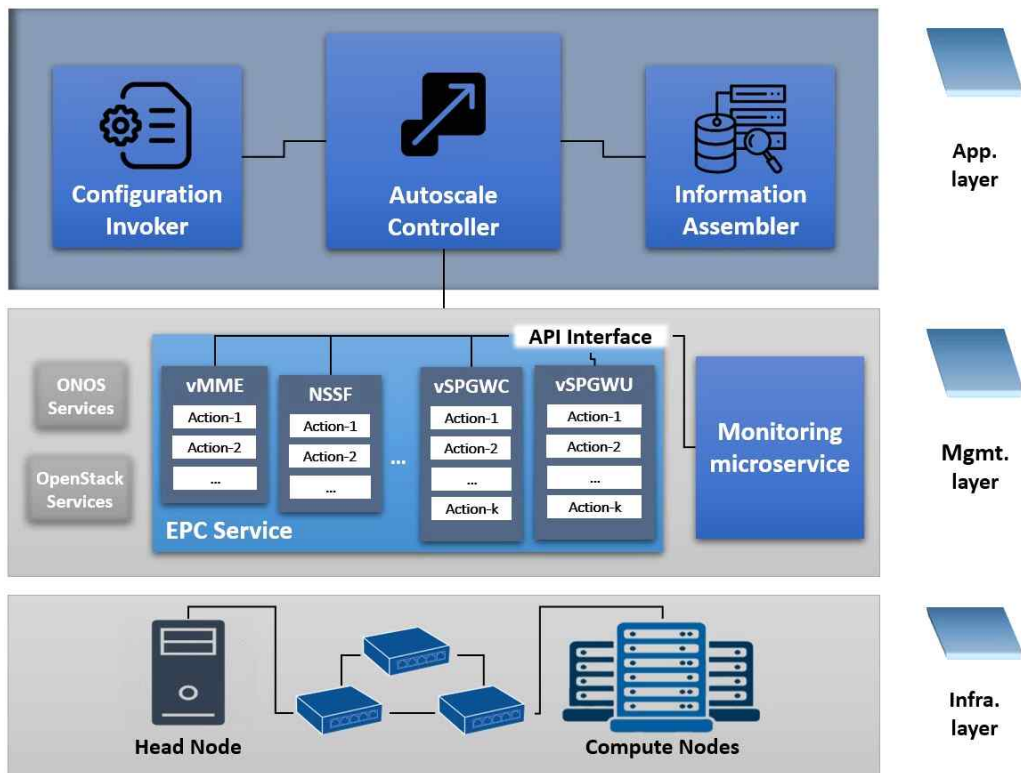


Figure 6 Overall Architecture for Autoscaling VNFs

Lets go into the details of each of the modules, their interfaces, the layers where they reside as well as the modifications proposed in the already used functionalities.

### 3.1 Modifications proposed

We have modified some of the parts inside the network services provided by OASIM [28,30]. These changes are made only for the purpose to enhance the functionality, performance wise as well as simpler architecture. The modifications made in the system are specifically related to the synchronizers.

First we modified each of the function of a VNF to dump the request time for that action. Then we exposed an API for each of the VNF's dumped data. All these changes were made to achieve simplicity, platform independency, customization and better performance. The most prominent benefit is the performance which we achieved by reducing the process communication involved in the cloud service based monitoring. So, we proposed a customized monitoring [4] solution that has less process latency as it is directly linked to the synchronizer level.

Another change made to the request dumping procedure is that, usually we store request-in time for the action [36], but we don't use request-finish time. as we have the execution-time for each of the action, thus the processing capabilities of the monitoring application were also enhance and optimized.

We also plan to enhance other parts to improve performance of the calculation of execution-times for the actions. The conversion of execution-times to clocks per cycle was a difficult job to perform. Though, the results were productive by achieving the defined criteria through the designed and programmed system, we plan to further enhance this mechanism of conversion to improve the system.

## 3.2 Layers

We have 3 layers in the proposed architecture built upon the M-CORD framework. We describe them as the Application, Management and Infrastructure layers. Lets get into the details of each of it

### 3.2.1 Infrastructure Layer

This part of the architecture includes is the infrastructure on which the cloud services run and collectively provide us an NFVI. OpenStack plays the part of VIM over here and for the purpose of NFVI, we have an hypervisor KVM/LIBVIRT [6]. It also includes virtual switches which lets the virtual machines connect to each other through networks which are created using the TOSCA [6,] formatted configuration.

This layer has three nodes named as corddev-node, head-node and compute-node. We can have more compute nodes as much as we can. But for our use-case, we only have one compute-node. The VNF(s) running on the virtual-machine as instances are inside the compute-nodes. These network services comprising of VNF services could be IMS [37], VoIP, Calling etc. While the management services and the synchronizers (element management services) run on the head-node. Apart from these services there are synchronizers running for each of the project service of OpenStack (cloud) as well as for ONOS (SDN). All this setup is done using the corddev-node. Once the head-node and compute-node are up, running and functional the corddev-node is of no use to us. The two main concerned parts of the infrastructure layer are head-node on which services run and the compute-node on which the actual VNF(s) run with the switched virtual networks. In the next section, we define the Management layer that plays an important role in the monitoring of resources in an efficient way.



### 3.2.2 Management Layer

This layer contains most of the part on head-node, which was told previously in detail that it holds the management. As we know that the OpenStack monitoring or any of the cloud supported monitoring systems are not supposed to be deployed onto a known location which does not let the overall system to measure the process latency or delay. For this purpose, we propose a monitoring microservice that is supposed to be on the head-node, closer to the other element management services. This gives the monitoring service a straight forward connection to the synchronizers resulting in less process latency, which was one of the goals in our architecture. Secondly it is a flexible and custom way to be adopted by the developers. Because of the messaging mechanism between the monitoring service and synchronizers is achieved by the exchange of JSON formatted messages, this lets developers to focus on what to develop rather than what platforms/languages to chose for development.

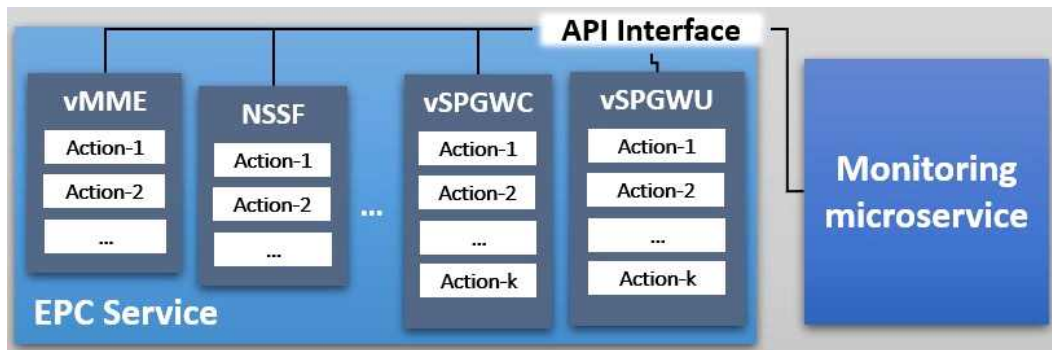


Figure 7 Monitoring microservice communication with synchronizers via an API

Now lets talk about the application's functionality. It fetches information from the synchronizers and passes it over to the Information Assembler residing inside the Autoscaling application at the application layer. The data includes the requests related to a specific action of a VNF. This information is stored in the Information Assembler's store about which we will further discuss about, in details under the section of Modules.

### 3.2.3 Application Layer

This part of the architecture includes our Autoscaling Application, which is the brain to decide. It is responsible to scale up/down the resources according to the usage of resources based on real-time data. Putting the Autoscaling Application on this layer is that the application deployable becomes easier to upgrade or migrate the application in this case. We put three modules as our contribution inside this application. The modules are Information Assembler, Autoscale Controller and the Configuration Invoker. The role of Information Assembler is to store the information of resource usage. It's design is highly customizable. The decisions made in this application are intelligent thus provide an efficient orchestration [9,38] mechanism

### 3.3 Modules

The system contains two modules and some changes in the modules of the proposed system. The first one is Intent [14] Based Autoscale application while the second one is the Monitoring [4] microservice and the other modules already in existence are the microservices based synchronizer which have been slightly modified to accomplish the criteria defined in the objectives of this document.

Before going into much details of each of the modules, I feel the necessity to explain how they interact with each other diagrammatically.

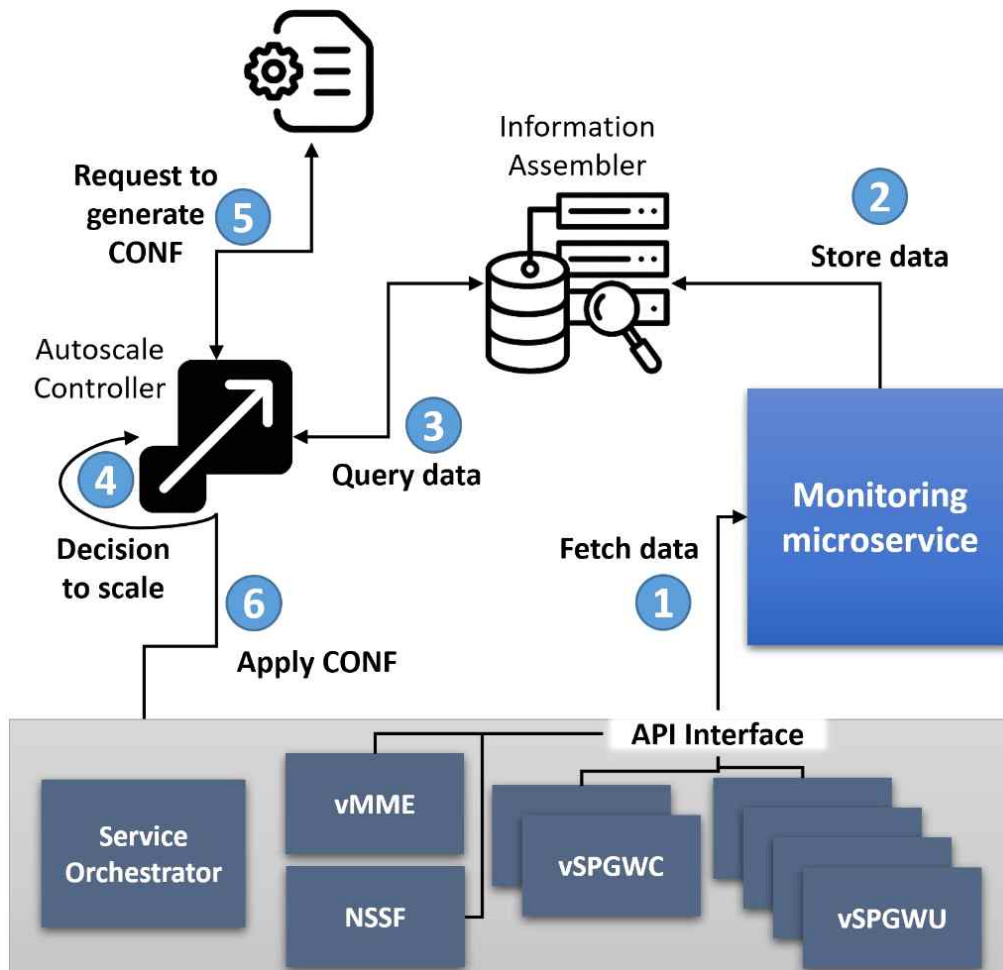


Figure 8 Steps - Overall mechanism between the system components

All these modules work in parallel and are not bound to execute their programs based on triggers. This provides a mechanism to achieve parallelism in computing of the management and application.

For the purpose of easy illustration, we believe to start the from synchronizers. We have modified them to fit in the situation and proposed scenario of ours to accomplish efficient mechanism. Whenever the requests are received by the VNF(s), the request with its time-stamp is dumped onto a storage residing on the head-node. This request is associated with a specific instance's action performed. Now coming towards the monitoring application, it fetches the information stored from each of the synchronizers (element managers). This fetching process takes place via API interfaces. The messaging format exchange between the monitoring application and the synchronizers is JSON. This allows the creation of a platform independent application as well as allows the application to be used on multiple type of platforms.

The monitoring microservice then stores this information onto the Information Assembler. The design for this store is provided in the next coming subsections. For database development, we used MySQL and the tables were named as VNFs, Instances, Actions and Requests. In simple words, these tables contain the request information for each of the VNF instance. It also stores the necessary information such as Network service catalogue with it instance catalogue as well. All this fetching procedure occurs on regular cycle of intervals, to maintain consistency in our proposed system which is responsibility of Autoscaling application to take decisions based on latest and real-time historical data.

The autoscaling [11] decision process is the key factor of our system i.e. Autoscale Controller. The factors that make it unique are the conversion of CPU(s) as clocks per cycle and using the execution-time [36,39] of a program to estimate the CPU(s) required for the next interval. Another important factor in

selecting the number of CPU(s) is the weight factor [11]. It is used for making comparisons between the nesting level of virtualization. The more the nested level increases, we decrease the weight as the virtualized CPU loses its capacity in worst cases. An algorithm for the purpose to illustrate decision-making process is going to be discussed in later section with diagrams and an algorithmic representation.

The last step to be taken in this mechanism (for one cycle of decision) is the generation of configurations. For this purpose, we've developed a module named as a Configuration Invoker. This module takes number of instances, their id(s) as an input, number of resources to be assigned. With the context of these inputs, this module generates a configuration in the TOSCA format, which is then passed backed as a response to the Autoscale Controller.

The Autoscale controller posts this configuration to the XOS via its exposed API(s). The rest of the responsibility is fulfilled by the services deployed on the head-node of our system deployment. The XOS synchronizers store this configuration information into the XOS-DB [20], while the other synchronizers run a scheduled check of change in the configurations. By detecting this change in the configuration, they try to reflect the changes to the VNF(s) deployed and running on the compute-nodes.

### 3.3.1 Intent Based Autoscale Application

This part of proposed is intended to explain the internals of Autoscale application. Now that we've briefly defined the modules, they need to be technically described in detail.

#### 3.3.1.1 Information Assembler

It is the catalogue where the information related to VNF(s), instances, database is stored. It has four tables named as VNFs, Instances, Actions and the Requests. As from the name VNFs, is obvious that it is the catalogue for VNFs. It holds the details of a VNF such as its name, number of actions it has in its program, minimum cps (clocks per cycle speed) required for the VNF to boot up and the threshold cps (clocks per cycle) required for the program to run up to its best performance. Instances table have the information related to an actual instance of VNF running. One VNF can have multiple instances. From the table Actions, we can presume that it should contain the number of actions that it can perform. So it indirectly to saying that this table contains the details of actions of an instances. It also has an execution-time [39] associated with each of the action it contains in its database. The Requests table, contains the requests received by a specific instance.

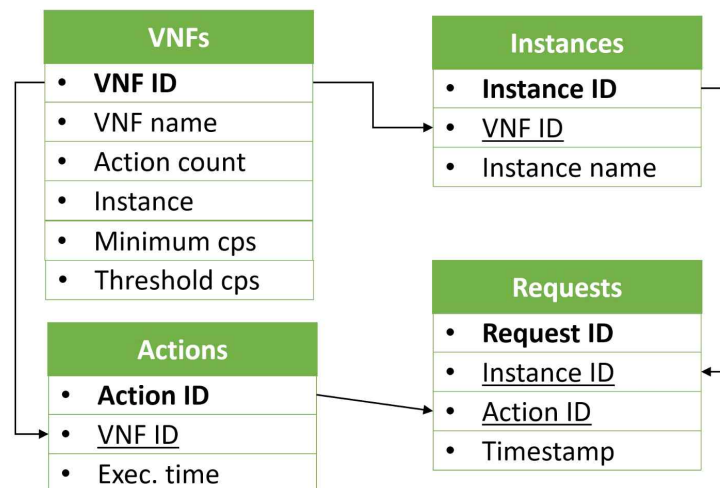


Figure 9 Information Assembler Database Tables

One thing to be noted is that we use different iterators throughout this document.  $i$ ,  $j$ ,  $k$  and  $x$ . “ $i$ ” relates to the VNFs list iterator, “ $j$ ” relates to the instance iterator, “ $k$ ” relates to the action iterator while “ $x$ ” is the iterator used for the number of requests received by a specific instance. Below are the sets of VNF(s) deployed and the set of requests received by an instance:

$$\begin{aligned} VNFs &= \{vnf_1, vnf_2, vnf_3, \dots, vnf_{n_i}\} \\ REQs &= \{req_1, req_2, req_3, \dots, req_{n_{jx}}\} \end{aligned}$$

IC denotes Instance count while AC denotes Action count. The below two equations are to accumulate the Instance count as well as the Actions count for a specific VNF and an instance respectively:

$$\begin{aligned} IC_i &= \sum_{j=1}^{n_j} (IC_{i_j}) \\ AC_i &= \sum_{k=1}^{n_k} (AC_{i_k}) \end{aligned}$$

The ET denotes to execution-time [39]. First equation shows the execution-time for a specific instance “ $i$ ”, by assuming 1 request per action of the instance. The second equation shows the overall execution-time for the specific interval of time “ $t$ ”:

$$\begin{aligned} ET_i &= \sum_{k=1}^{n_k} (ET_{i_k}) \\ ET_{j_t} &= \sum_{k=1}^{n_k} (ET_k * x) \end{aligned}$$

### 3.3.1.2 Autoscale Controller

First of all there are some norms to be standardized before moving on further. Lets consider PR is the sorted set of VNFs in order of priority. This priority shows the priority of necessity to be auto-scaled first:

$$PR = \{ vnf_3, vnf_n, \dots vnf_2 \}$$

The procedure of Autoscaling Controller flow-chart is as follows. First of all it iterates the VNFs list by priority. Then loops on each of the instance and the inner dark-grey box use the inner iteration, specific to an instance. First it fetches the necessary information from the Information Assembler, then calculates cps used for this interval and estimate the cps required for the next interval. This calculation of cps from execution-time is done with the help of getting information about the number of requests per second.

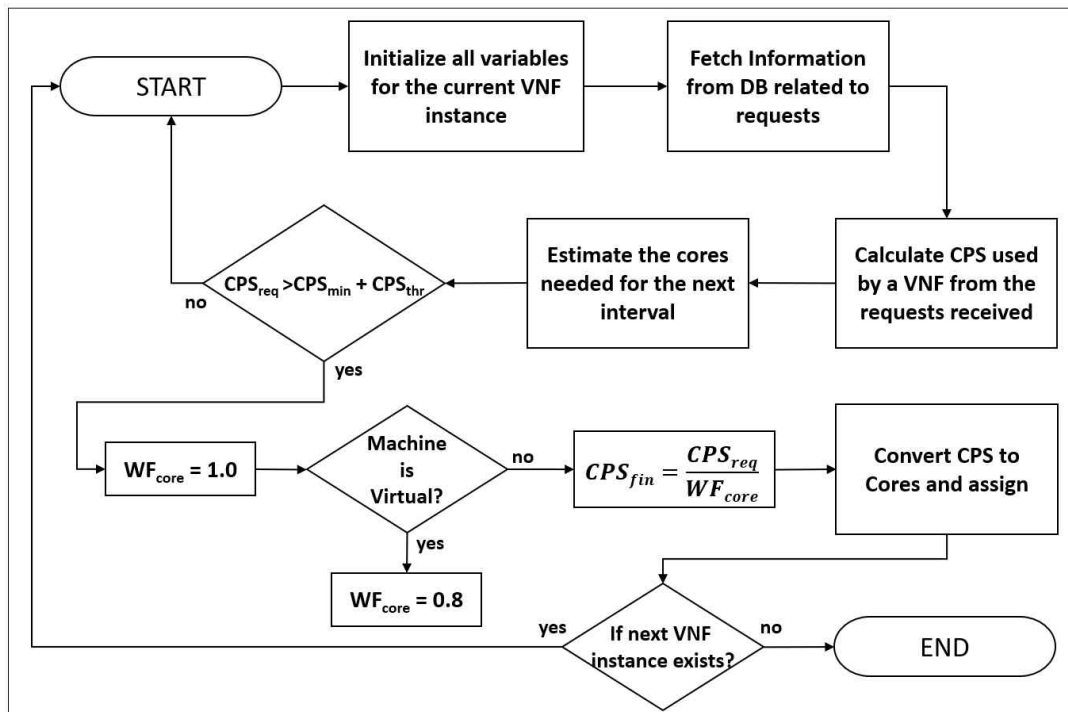


Figure 10 Autoscale-Controller - Flow chart of decision-making process



We've pre-calculated the minimum cps and threshold cps required for the current running instance. If the required cps (calculated) is greater than sum of minimum cps and threshold cps, then it moves on otherwise the algorithm continues with the same steps for next instance in cycle. While we're assuming here that check of assuring the resource is passed so the algorithm will assign a weight factor [11] on the basis of checking whether resource is physical or virtual. By default, it is 1.0, while for virtual resource its value turns out to be 0.8 that shows the performance is low in worst cases for virtual cores.

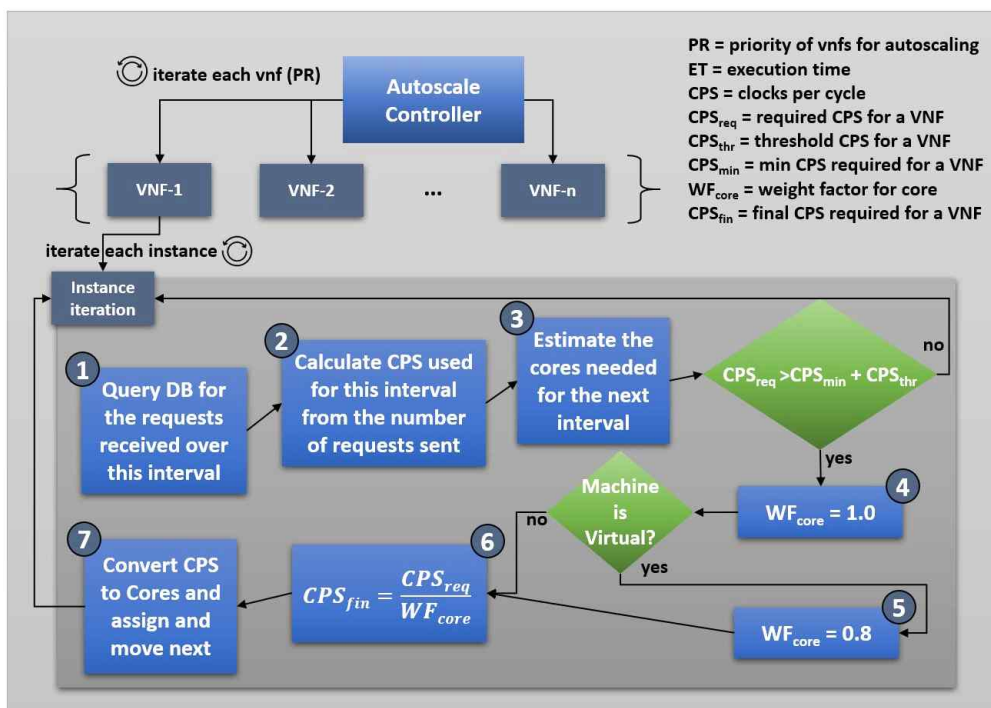


Figure 11 Autoscale-Controller - Mechanism of decision-making process

Then by using required cps (that was calculated in the previous steps) and weight factor for core, we put variation in core clocks per cycle estimated. The final goal is achieved by multiplying final cps required with factor of  $10^9$ . After finalizing value for each instance, autoscale controller requests Configuration Invoker to generate TOSCA configuration. This way it receives the configuration and applies it through a post request to XOS (the Service Orchestrator).

Now we explain the same mechanism, in the form of an algorithm. The first line shows the set of VNFs in priorities. It loops each VNF and then each of its instances. It queries the results of that specific instance for the requests received in that specific interval of time. Calculates the required cps for this instance. Fetches the minimum cps required in order to boot the VNF instance while considering the threshold cps value that ensures the minimum extra processing to be assigned. This process is followed by the check to ensure the correct estimation of cores to be assigned. It assigns a weight factor for the nesting level of virtualization used. At the last part of this algorithm, it converts the cps to GHz unit to check how much frequency of processing is required.

For the assignation of RAM, we use the defined standard which states that the ratio between a CORE and RAM should 1:4. We modified it. So, considering an example of 4 cores to be assigned to a VNF, We use the weight factor of assigning the RAM 0.4 for all of the VNFs. and take the ceiling of that value.

---

```

1:  $PR = \{ vnf_3, vnf_n, \dots vnf_2 \}$ 
2: for all  $vnf$  in  $PR$  do
3:   for all  $instance$  of  $vnf$  do
4:     initialize  $cps_{fin} = null$ 
5:     query requests for  $instance$ 
6:     calculate  $cps_{req}$  for next interval
7:     setup the  $cps_{min}$  and  $cps_{thr}$  for this  $instance$ 
8:     if  $cps_{req} > (cps_{min} + cps_{thr})$  then
9:        $wf_{core} = 1.0$ 
10:      if  $machine$  is virtual then
11:         $wf_{core} = 0.8$ 
12:      end if
13:       $cps_{fin} = \frac{cps_{req}}{wf_{core}}$ 
14:    end if
15:     $GHz_{req} = cps_{fin} * 10^9$ 
16:  end for
17: end for

```

---

Figure 12 Autoscale-Controller - Algorithm of decision-making

### 3.3.1.3 Configuration Invoker

This module of our Autoscaling application generates the TOSCA by translating the JSON requirements sent by the Autoscale Controller. It has a few file formats already stored into the CORD directory, from where it writes the file and replaces the number of cores and RAM parameters for each of the VNF. As the number of VNFs as the number of instances with the number of cores will also change so, we also have to put the number of instance in the TOSCA format. So, it results in creation or upgradation of resources.

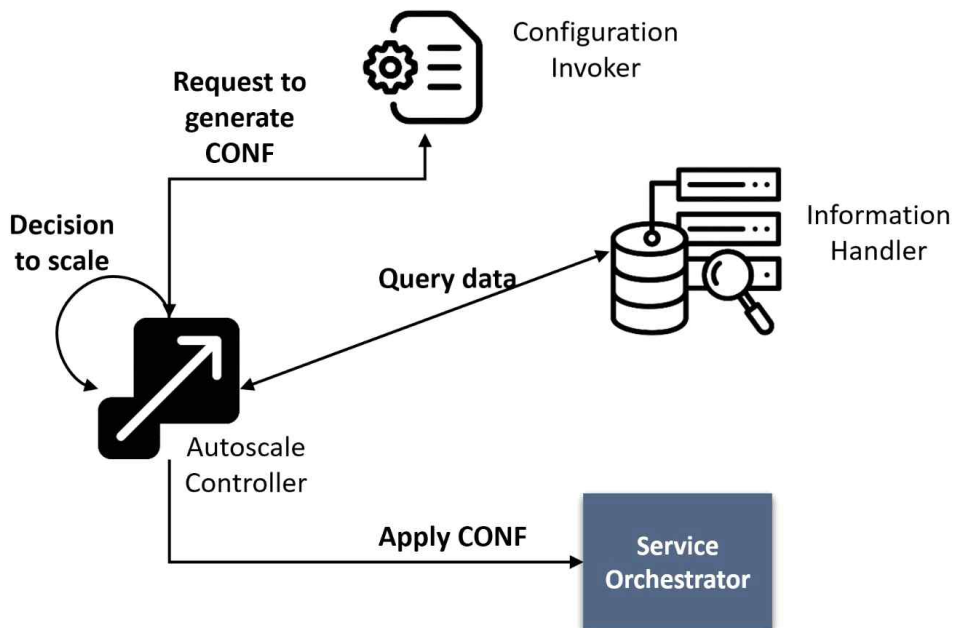


Figure 13 Configuration Invoker Communication with Autoscale Controller

### 3.3.2 Monitoring Microservice

The microservice is a container based application on management layer. It resides besides the other synchronizer services running on the head-node of our configured system. It interacts with the synchronizers specific to a VNF through a well defined API. We have modified the synchronizers to dump the real-time data on the current node for a specific interval of time. This data is then fetched by the monitoring microservice application from the synchronizers and stores it as an object. This object is handed over to the Autoscale application. The responsible module of Autoscale application for storing this information is the Information Assembler. Once the information is stored, then the responsibility of Monitoring microservice is finished until the time for next cycle of fetching comes.

Monitoring microservice was developed in Python and libraries were used to read and write monitoring data. All this mechanism was only to support a custom monitoring solution, which is not platform dependent. The developer and the organization who are providing this solution do not need to worry about the platform as the message communication with the synchronizers is API based. This property gives us platform independency. It also gives the application developers to deploy their application on multiple platforms with a slighter modification in the mechanism procedures.

### 3.4 Configuration of System

We developed and tested our proposed system on a server Intel Xeon with 16 physical cores having the capacity to handle instructions at the frequency of 2.4 GHz. Hyperthreading level was 2, which means we had 32 virtual cores. It had 64 GB of memory (RAM) and 3.9 TB of the Hard disk.

For the deployment part, we used the CORD platform with M-CORD profile configured with OAISIM Network services. First we configured the deployment of network services one by one. After building it the final result was to have three nodes. corddev, head and compute. As the name suggests that corddev node was used for the purpose to setup head and compute node. Specification for each of the nodes are provided in the figure below. Hyperthreading on the compute node was done through the configurations at start, to increase number of the cores. So, this was the overall configurations.

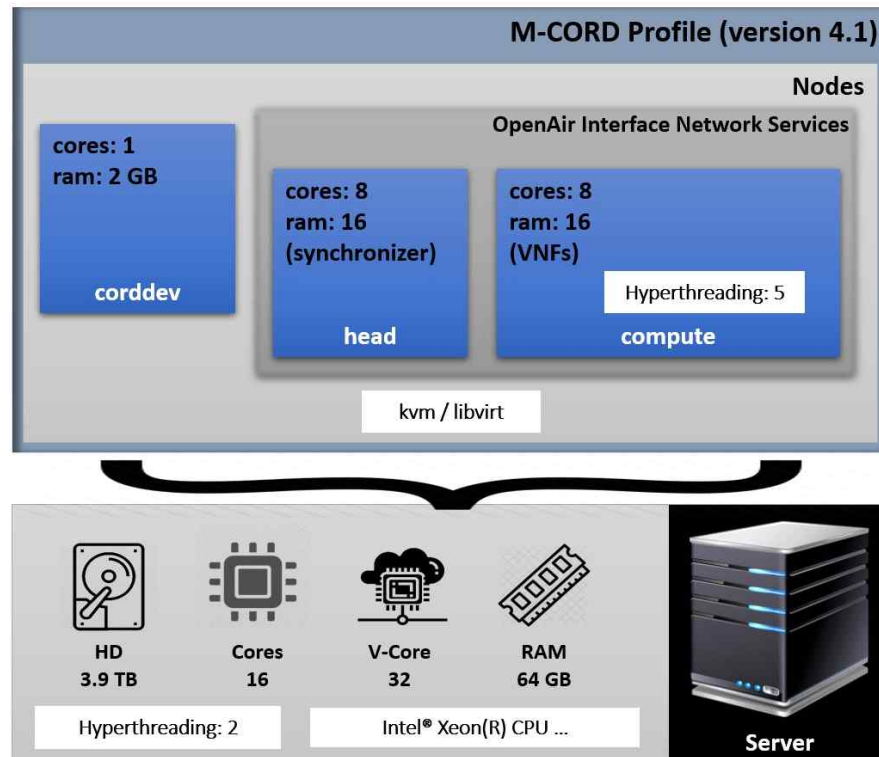


Figure 14 Configuration and Requirements of the System

## 3.5 Specifications

This section provides the specifications of the application and microservice that we proposed and developed. The content is divided into two subsections to enlighten the technical specifications and messages exchanged from each of the modules with other modules. The subparts are as follows:

### 3.5.1 Application specifications

We have built our Autoscaling application in Python. All the mechanism involved it is purely based on logic the factors involved, to autoscale the VNFs. This application has three modules named as Autoscale Controller for the purpose to decide when the system VNFs will scale up/down depending upon on the usage of the resources. These resources are CPU usage and RAM usage.

Then comes the Information Assembler. We used the MySQL database platform for the database creation. The data is stored on it by the monitoring application and the maintenance of this data is done by the Information Assembler. The usage of this data is requested by the Autoscale Controller.

### 3.5.2 Microservice specifications

The Microservice application is also developed in Python. It is implemented as a container application which can interact with the other synchronizers through its exposed interfaces. It is deployed in parallel with the synchronizers. These synchronizers are often termed as Elementary Management Services as they are involved in the management of VNF's functionality rather than the management of VNFs lifecycle which is the job of a VNFM. Each VNF's design is modified such that it exposes the information of incoming requests.

Microservice application communicates with the Information Assembler to post the data over there. Thus the purpose of Microservice application is to monitor the real-time data and pass it onto the Autoscale application.

## 4 Evaluation and Results

This section contains the requirements for the environment setup, evaluation metrics to be considered and then the evaluation results. The below table describes the detailed specifications used for the physical system that we used. Afterwards, it puts highlights the deployment details to accomplish the test-environment. The table is as follows:

Physical System	Server Details	Intel Xeon(R) CPU ...
	Server Hard Drive	3.9 TB
	System Cores	16 (2.4 GHz)
	Hyperthreads	2
	Virtual Cores	16*(2) = 32
	NIC	2
	System Memory	64 GB
	Operating System	Ubuntu 16.04.3 LTS
-----		
Deployment System	CORD profile	M-CORD
	Network Services	OpenAir Interface
	M-CORD release	version 4.1
	Nodes	corddev, head, compute
	corddev CORES/RAM	1/2
	head CORES/RAM	8/16
	compute CORES/RAM	8/16
	compute hyperthreads	5
	compute node virtual cores	8*(5) = 40
	Database type	MySQL
	Python plugin for database	SqlAlchemy
	Languages	Python 3.6
	Development IDE	PyCharm 2019.1.1
Hypervisor	kvm/libvirt	

Table 1 Specifications of Physical and Deployment system

## 4.1 CORD Configuration Steps

First we start by giving a brief overview of the targets to consider while setting up the deployment. The figure shown below shows steps for the installation to complete. These steps are categorized into 7 sections.

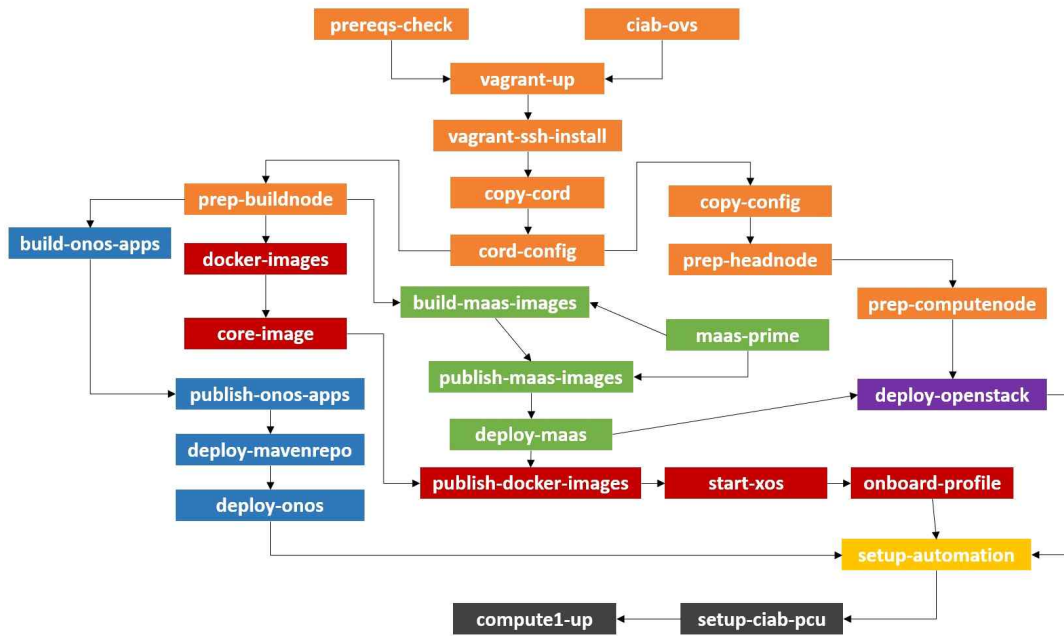


Figure 15 CORD installation steps

This section contains the steps to be taken to deploy the M-CORD framework.

### 4.1.1 Preparation-targets

The orange boxes shown refer to the preparation targets. These steps are the initial steps to setup the head-node which contains synchronizers. It also builds the VMs before the configuration of these VMs take place.

### 4.1.2 MaaS-targets

This term stands for Metal as a Service and allows the operator to use the physical machine as a virtual machine. It helps the system to treat a physical resource as a virtual resource. Green boxes represent these targets in figure.



#### 4.1.3 XOS-targets

The targets represented as red boxes are responsible to pull the images, build them and then to run the images. These images are run as Docker-containers. One more purpose of these targets is to configure the XOS.

#### 4.1.4 ONOS-targets

The blue boxes in previous figure represent ONOS targets. These targets install the instances of ONOS as well as the applications to be used into them. Most popular ONOS applications used under this platform are CORDVTN and vRouter.

#### 4.1.5 OpenStack-targets

The only one target that deploys OpenStack is `deploy-openstack` highlighted as purple box. This target is divided into two main steps. First one is to create and configure the LXD containers for each subsystem of the OpenStack cloud platform. Secondly, it deploys OpenStack with the help of Juju which is an orchestration tool used to deploy, manage the applications running on cloud.

#### 4.1.6 Post Onboarding-targets

This target configures the compute-node by using the playbook named as `cord-automation-playbook.yml`. The specific target linked to this playbook is termed as `setup-automation`. The two generated configurations are further used to verify the compute-node's deployment.

#### 4.1.7 Additional CiaB-targets

This step is specific for the CiaB deployment. Here we just mention the two targets which are `setup-ciab-pcu` and `compute1-up`. The first mentioned target named as `setup-ciab-pcu` allows the remote control of power for the CiaB. `Compute1-up` target boots up the compute node.

## 4.2 Evaluation Metrics

We used CPU usage to evaluate the proposed and developed system. As the resource of CPU is critical and needs to be estimated as accurate as possible to neither over-utilize nor under-utilize the CPU resource usage. We show the CPU percentage used for the assigned CPU per VNFs, per all of the VNFs. We also show the evaluations for the overall CPU resources used and freed up for the tenant.

### 4.2.1 Assigned CPU Usage

This subsection highlights what the assigned cpu usage means. When we use the term “assigned CPU usage”, we are talking about the number of CPU percentage used by the entity/entities that had been assigned by the orchestrator. Entity/entities refers to that we evaluate the metric assigned cpu usage for each VNF instance as well as for the overall VNFs building up a single network service.

### 4.2.2 Overall CPU Usage

We use this metric in order to show the total number of resources that are freed up which helps out the orchestrator to verify and confirm the number of resources freed up that can be used for other network domains. Our results show the number of resources are freed up on one side and on the other side prove that the resources assigned to them are being used up to the optimum.

### 4.3 Evaluation Results

This section provides the graphical and tabular representations for the results that we concluded from our experiments. For the purpose of demonstration, we divided this part into two subparts as follows:

#### 4.3.1 Assigned CPU Usage

We divided this section into two subparts as follows:

##### 4.3.1.1 Average stats for each VNF

The table shown below, contains the statistical details of the CPU percentage used for each of the VNF instance running during this 't' time-interval. By analysing the statistics, we can clearly see that the number of resources are being assigned as per needed. This causes the system to free up the unused resources. It also ensures the optimum usage of resources during the 't' time-interval. At start, most of the VNF instance are using the assigned CPU(s) from the range 30 to 45. But after running 7 cycle intervals of time 't', we are able to get better results that use 50 to 60 percentage of the CPU assigned CPU. It is deduced from the explanation of below shown statistics that our system is moving towards efficient usage of resources. The stats for cycle # 1 are as follows:

VNF	Usage % at time interval 't' for VNF for first cycle						
	t=1	t=2	t=3	t=4	t=5	t=6	t=7
eNB	31.25	46.00	76.00	51.67	58.00	60.00	60.00
vMME	33.75	50.00	78.00	53.33	62.00	60.00	58.00
NSSF	50.00	50.00	65.00	45.00	35.00	40.00	50.00
vHSS	50.00	60.00	70.00	26.67	40.00	50.00	45.00
vSPGWC	57.50	56.67	75.00	52.50	60.00	46.67	43.33
vSPGWU	43.75	53.33	70.00	58.00	44.00	46.00	46.00

Table 2 Cycle 1: Usage percentage for each VNF at time interval 't'

Now we present the graphical form of statistical data shown in above table. It can be seen that the all of the VNFs are using almost 50-60 of the assigned resources. The number of free resources that not need to be assigned are also released. This way the resources are accommodated according to the situation and the equal distribution of the resources is achieved. This mechanism concludes by the this graph that the system is assigning the resources up to optimum value. Hence, the resources are neither over-utilized nor under-utilized. The graphical stats for cycle # 1 are as follows:

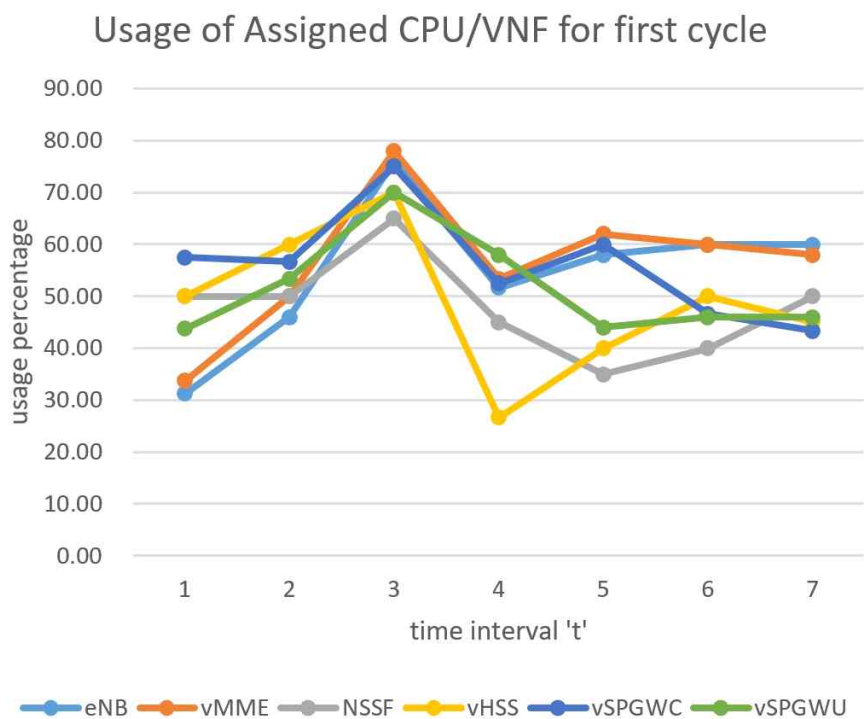


Figure 16 Usage of Assigned CPU/VNF for first cycle

We performed the same test on different time-slots from the start to check the variation in our statistics for the second time. The stats for cycle # 2 are as follows:

VNF	Usage % at time interval 't' for VNF for first cycle						
	t=1	t=2	t=3	t=4	t=5	t=6	t=7
eNB	30.00	48.00	58.00	51.67	60.00	60.00	64.00
vMME	31.25	52.00	76.00	53.33	68.00	58.00	66.00
NSSF	45.00	55.00	65.00	50.00	30.00	40.00	60.00
vHSS	40.00	55.00	70.00	30.00	45.00	55.00	35.00
vSPGWC	55.00	63.33	75.00	52.50	63.33	46.67	40.00
vSPGWU	42.50	55.00	68.33	60.00	46.00	42.00	48.00

Table 3 Cycle 2: Usage percentage for each VNF at time interval 't'

For the graph given above we depict these values to be shown in the below graph. The graphical stats for the second cycle are as follows:

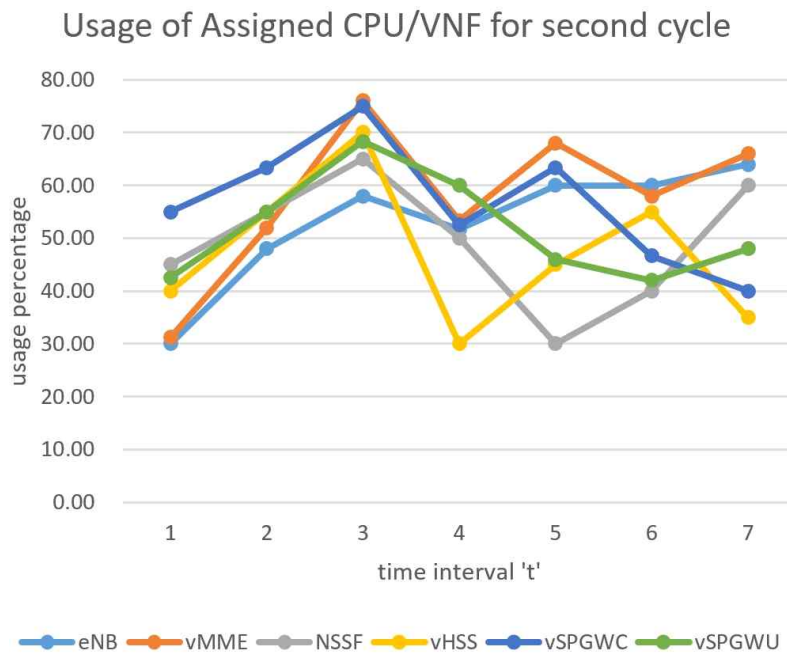


Figure 17 Usage of Assigned CPU/VNF for second cycle

We performed the same test on different time-slots from the start to check the variation in our statistics for the third time. The stats for cycle # 3 are as follows:

VNF	Usage % at time interval 't' for VNF for first cycle						
	t=1	t=2	t=3	t=4	t=5	t=6	t=7
eNB	26.25	42.00	54.00	50.00	62.00	60.00	66.00
vMME	33.75	50.00	66.00	58.33	68.00	54.00	64.00
NSSF	40.00	60.00	45.00	45.00	40.00	35.00	50.00
vHSS	35.00	50.00	55.00	33.33	45.00	35.00	50.00
vSPGWC	50.00	50.00	67.50	60.00	60.00	43.33	50.00
vSPGWU	40.00	46.67	65.00	68.00	46.00	44.00	48.00

Table 4 Cycle 3: Usage percentage for each VNF at time interval 't'

For the graph given above we depict these values to be shown in the below graph. The graphical stats for the third cycle are as follows:

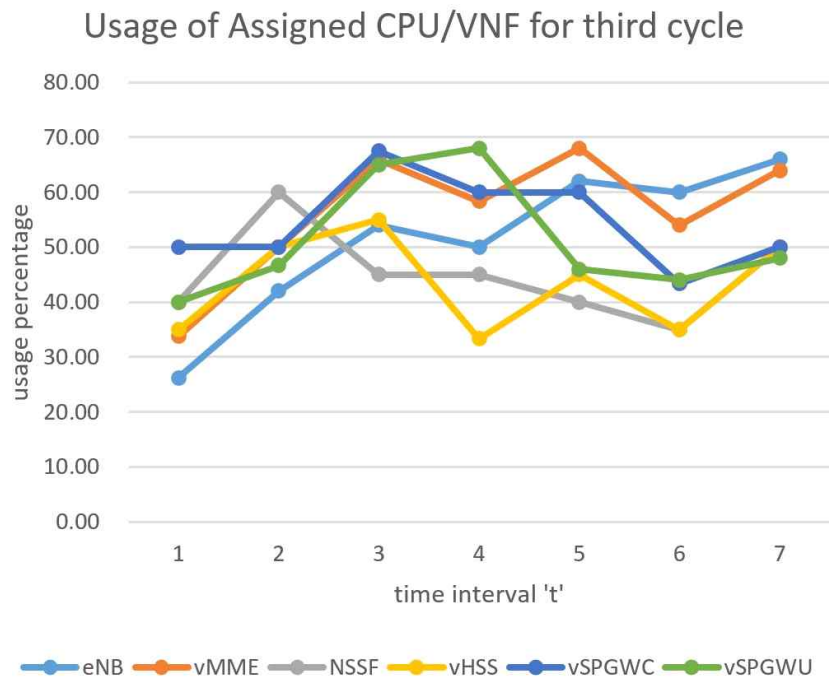


Figure 18 Usage of Assigned CPU/VNF for third cycle

#### 4.3.1.2 Average stats for all VNFs

The below table, contains the statistical details of the CPU percentage used for overall VNF instances running during the 't' time-interval. By the deep analysis of these statistics, it is clear that the number of resources are being freed up. This ensures that the resources which are being assigned are properly being utilized. The percentage usage of overall VNFs has moved the range 40 percent to 51, referring to the efficient usage of resources. This also ensures freeing up the resources. It can be deduced from the explanation that the system is using the resources in an efficient manner.

Time	Usage % at time interval 't' for all VNFs			
	Assigned	Assign used	Used %	Available %
t=1	32	13	40.63	59.38
t=2	23	12	51.74	48.26
t=3	24	18	73.33	26.67
t=4	26	13	50.00	50.00
t=5	22	12	52.27	47.73
t=6	22	12	52.27	47.73
t=7	22	11	51.82	48.18

Table 5 Cycle 1: Usage percentage for all VNFs at time interval 't'

Now we show the graphical form for the stats discussed above mentioned in the previous tabular form. It is shown that in the first cycle of time-interval 't', the usage of assigned resources is almost 41%, which shows that the resources assigned to it should have been different so that almost 50-60 of the resources were used. But as this is this the first cycle so after some time of autoscaling cycles, our system will take care of this problem automatically. The percentage usage of overall VNFs has moved the range 40 percent to 51, which supports the claim/statement that the resources usage has been increased by the percentage it uses. This technique frees the resources that are not being used by the VNFs and creates an healthy environment for other tenants to use the underlying resources provided by the virtualization layer whether it be via the hypervisor of compute, network or storage resource.

Usage of Assigned CPU/All VNF(s) for first cycle

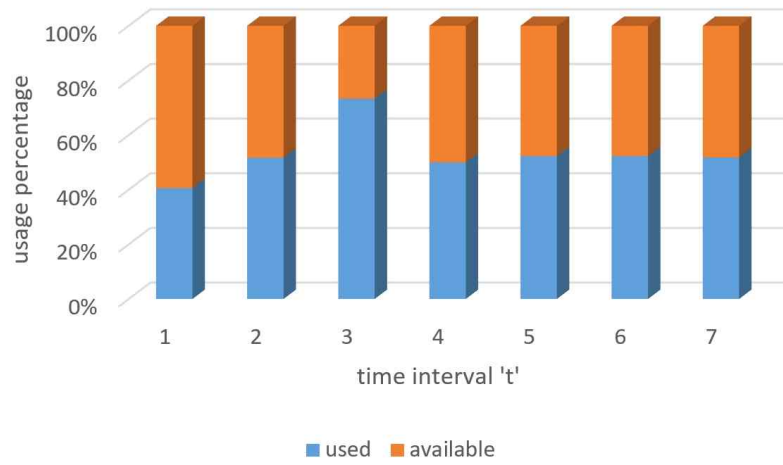


Figure 19 Usage of Assigned CPU/VNFs for first cycle



We performed the same test on different time-slots from the start to check the variation in our statistics for the second time. The below shown stats are for cycle # 2 and are as follows:

Time	Usage % at time interval 't' for all VNFs			
	Assigned	Assign used	Used %	Available %
t=1	32	12	38.13	61.88
t=2	23	12	53.91	46.09
t=3	24	17	68.75	31.25
t=4	26	13	51.15	48.85
t=5	22	12	55.00	45.00
t=6	22	11	51.36	48.64
t=7	22	12	54.55	45.45

Table 6 Cycle 2: Usage percentage for all VNFs at time interval 't'

The graph given below depicts the values from the above table. The graphical stats for the second cycle are as follows:

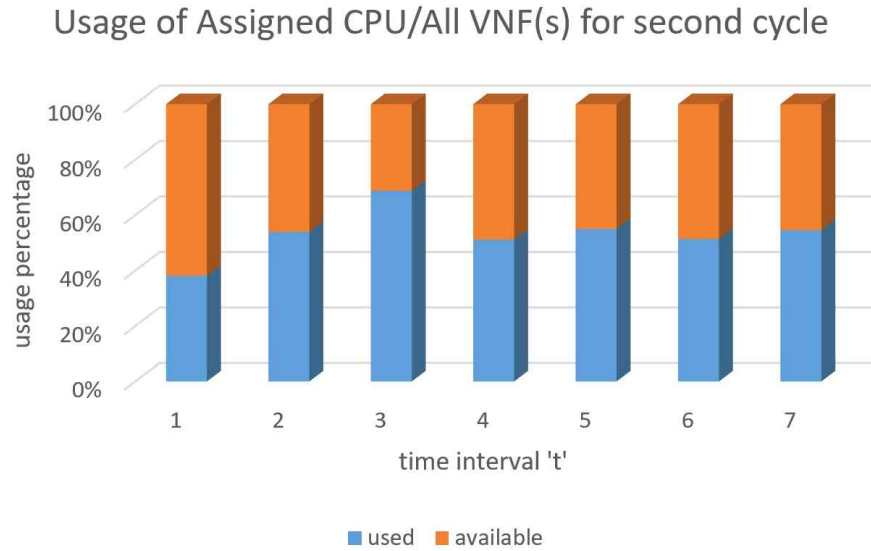


Figure 20 Usage of Assigned CPU/VNFs for second cycle

We performed the same test on different time-slots from the start to check the variation in our statistics for the third time. The below shown stats are for cycle # 3 and are as follows:

Time	Usage % at time interval 't' for all VNFs			
	Assigned	Assign used	Used %	Available %
t=1	32	12	35.94	64.06
t=2	23	11	48.26	51.74
t=3	24	15	60.83	39.17
t=4	26	14	54.62	45.38
t=5	22	12	55.91	44.09
t=6	22	11	48.18	51.82
t=7	22	12	56.36	43.64

Table 7 Cycle 3: Usage percentage for all the VNF at time interval 't'

The graph given below depicts the values from the above table. The graphical stats for the third cycle are as follows:

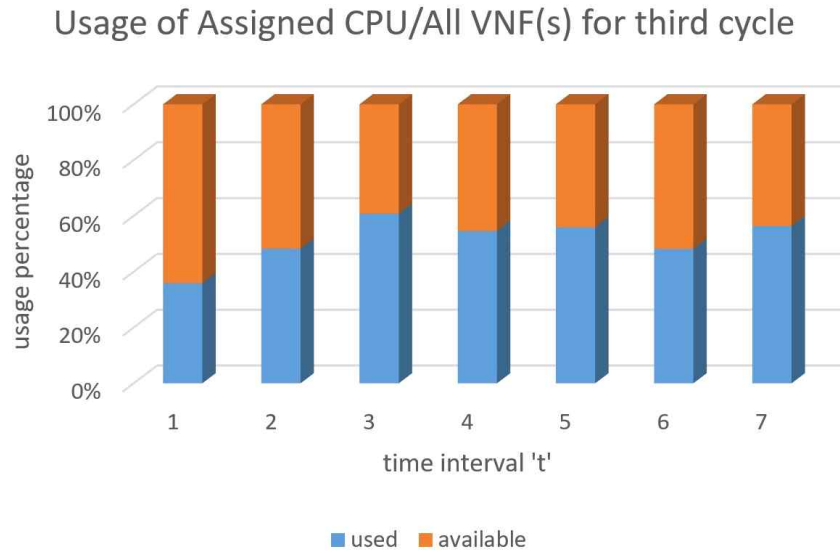


Figure 21 Usage of Assigned CPU/VNFs for third cycle

### 4.3.2 Overall CPU Usage

In this section we provide the overall CPU usage for all of the three cycles, each having seven intervals of time for autoscaling. Then we present the average values in the graphical form to show the consistency of our system behaviour by showing three cycles of procedural test.

#### 4.3.2.1 Average stats for overall CPU

The table contains the total number of CPU(s) used and available. These statistics are for overall VNF instances running during the 't' time-interval under different tenants control. But in our case, we have just considered a single tenant. The analysis of these statistics shows that at the start of the cycle for time-interval t=0, we assigned all of the 32 cores to all of the VNFs. By further analysing, we conclude that as the time-interval moves on, the free number of resources is increased by a valuable factor. This graph This ensures that the resources which are being assigned are properly being utilized. The percentage usage of overall VNFs has moved the range 40 percent to 51, referring to the efficient usage of resources. This also ensures freeing up the resources. It can be deduced from the explanation that the system is using the resources in an efficient manner.

Time	Usage % at time interval 't' for total CPU				
	Total	Total used	Total free	Used %	Free %
t=1	32	32	0	100.00	0.00
t=2	32	23	9	71.88	28.13
t=3	32	24	8	75.00	25.00
t=4	32	26	6	81.25	18.75
t=5	32	22	10	68.75	31.25
t=6	32	22	10	68.75	31.25
t=7	32	22	10	68.75	31.25

Table 8 Average usage percentage of Total CPU at time-interval 't'

The graph contains the information that shows the total number of CPU(s) being used and available for over the time-intervals. The free number of resources was 0% at start and after running 6-7 cycles of autoscaling time-intervals, The free number of resources that were assigned unnecessarily were up to 31.25%. These stats show the positive impact of autoscaling the resources efficiently. Hence these resources are freed up and can be used by other network services by the same tenant or other tenants sharing the same physical infrastructure via isolated virtual environments.

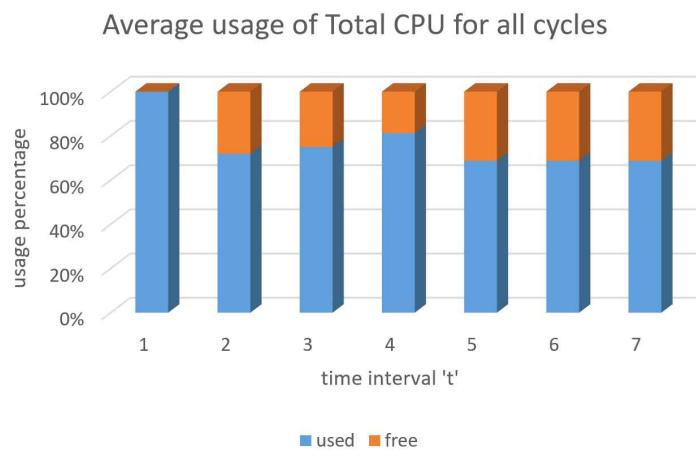


Figure 22 Usage of Total CPU (bar)

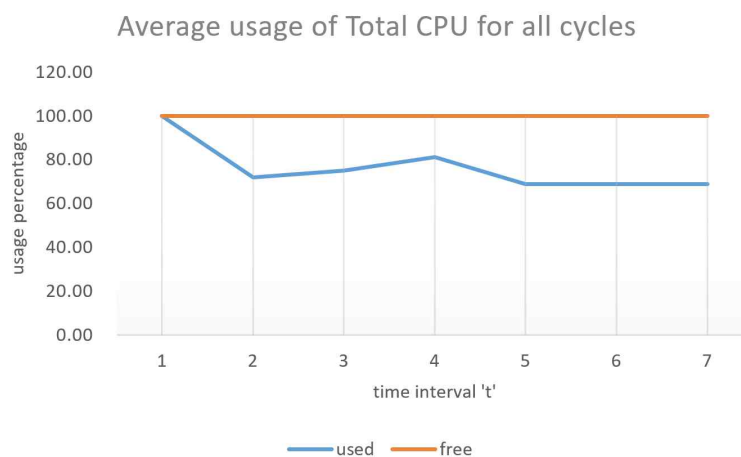


Figure 23 Usage of Total CPU (line)

This graph shown in Fig. 24. is taken from the paper [35] which shows the core allocation by applying different workloads on the virtual machine:

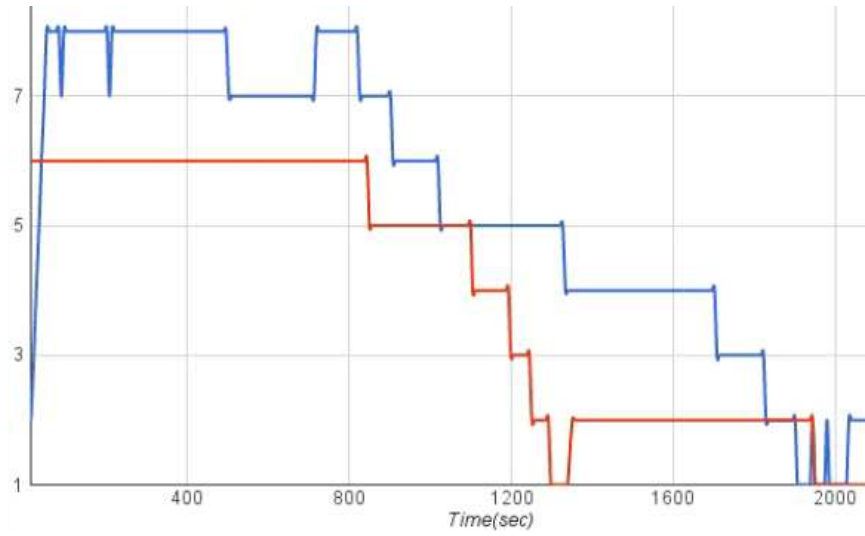


Figure 24 Core allocation by applying different workloads on VM

## 5 Conclusion and Future Work

The concluding remark of this system is that autoscaling [1] is a very important part of MANO [1,19] architecture, specifically in the VNFM (Virtual Network Function Management). The management should rely on the real-time data as well as the decision should be made intelligently. So, our system goal was achieved through the use of different factors such as the clocks per cycle, execution-times [36,39] of actions rather than the programs and weight-factors [11] involved for the involved virtualization-level. By combining all these factors into the decision-making algorithm, we introduced a system that makes sure that usage of resources is neither over-utilized nor under-utilized. The results at the end proved the efficient usage of resources such as CPU and RAM. For this purpose, we provide the results in the form of statistics and graphs. We used these results as a promising solution that distributes the resources equally. Another important attribute of our system is the platform independency. Our proposed and developed application and services communicate with each other, using the restful APIs as a channel. The communication done has been improved by introducing proper error responses between the services, so that the state or problems are easily identifiable.

We further plan to improve the system by introducing other factors involved in the network as well as to create a system that calculates the execution-time more accurately and precisely. Through the results were good, in this case but there is always a room for improvement. The conversion process of execution-times to clocks per cycle could be improved based on other factors that were not considered in this thesis. By researching and putting more efforts cloud yield better results in this field, as the nature of network-environments is too much diverse than it looks.

## References

1. William Sales, Emanuel Coutinho, and Jose Neuman de Souza “Auto-Scaling in NFV Using Tacker” 5th International Workshop on ADVANCES in ICT Infrastructures and Services, 2015.
2. Tulja Vamshi Kiran Bayakar, Anil Kumar Rangiseti, Antony Franklin A, Bheemarjuna Reddy Tamma “Auto Scaling of Data Plane VNFs in 5G Networks”, 13th International Conference on Network and Service Management (CNSM), 2017.
3. SDxCentral, “Understanding the SDN Architecture - SDN Control Plane & SDN Data Plane” [Online: accessed 06-2019]  
<https://www.sdxcentral.com/networking/sdn/definitions/inside-sdn-architecture>
4. Hynsik Yang, Briytone Mutichiro, Younghan Kim “Implementation of VNFC Monitoring Driver in the NFV Architecture”, International Conference on Information and Communication Technology Convergence (ICTC), 2017.
5. Asif Mehmood, Wang-Cheol Song “Autoscaling application and microservice based monitoring for VNFs”, KNOM Review (Proceeding of KNOM Conference 2019).
6. Nestor Bonjorn Lopez Ferran Canellas Cruz, “VIM Adaptation Layer for CORD”, 63 M.Sc. Thesis, Technical University of Denmark, Kgs. Lyngby, Denmark, 2018.
7. Open Cord, “M-CORD Project Overview” [Online: accessed 06-2019]  
<https://wiki.opencord.org/display/CORD/M-CORD+Project+Overview>
8. Muhammad Tahir Abbas, Talha Ahmed Khan, Asif Mahmood, Javier Jose Diaz Rivera, Wang-Cheol Song “Introducing network slice management inside M-CORD-based-5G framework”, 2018 IEEE/IFIP Network Operations and Management Symposium (NOMS), 2018.
9. B. Martini, M. Gharbaoui, S. Fichera, and P. Castoldi “Network Orchestration in Reliable 5G/NFV/SDN Infrastructures”, 19th International Conference on Transparent Optical Networks (ICTON), 2017.
10. Farah Fargo, Cihan Tunc, Youssif Al-Nashif, Ali Akoglu, Salim Hariri “Autonomic Workload and Resource Management of Cloud Computing Services”, International Conference on Cloud and Autonomic Computing, 2014.
11. Yi Ren, Tuan Phung-Duc, Jyh-Cheng Chen, and Zheng-Wei Yu “Dynamic Autoscaling Algorithm (DASA) for 5G Networks”, IEEE Global Communications Conference (GLOBECOM), 2016.

12. MuleSoft, “What are APIs and how do APIs work?” [Online: accessed 06-2019]  
<https://blogs.mulesoft.com/biz/tech-ramblings-biz/what-are-apis-how-do-apis-work>
13. Gaurav Goyal, Karanjit Singh, Dr. K.R. Ramkumar “A detailed analysis of data consistency concepts in data exchange formats (JSON & XML)”, International Conference on Computing, Communication and Automation (ICCCA), 2017.
14. Asif Mehmood, Talha Ahmed Khan, Javier Diaz Rivera, Wang-Cheol Song “An intent-based mechanism to create a network slice using contracts”, Proceedings of Symposium of the Korean Institute of communications and Information Sciences, 2018.
15. OpenStack, “What is OpenStack?” [Online: accessed 06-2019]  
<https://www.openstack.org/software>
16. Rajendra Chayapathi, Syed Farrukh Hassan, Paresh Shah “Network Functions Virtualization (NFV) with a Touch of SDN”, A book published by Addison-Wesley.
17. ETSI, “ETSI GS NFV 002 V1.2.1” [Online: accessed 06-2019]  
[https://www.etsi.org/deliver/etsi\\_gs/NFV/001\\_099/002/01.02.01\\_60/gs\\_NFV002v010201p.pdf](https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf)
18. SDxCentral, “What is OPNFV or Open Platform for NFV Project?” [Online: accessed 06-2019] <https://www.sdxcentral.com/networking/nfv/definitions/opnfv>
19. ETSI, “ETSI GS NFV-MAN 001 V1.1.1” [Online: accessed 06-2019]  
[https://www.etsi.org/deliver/etsi\\_gs/NFV-MAN/001\\_099/001/01.01.01\\_60/gs\\_nfv-man001v010101p.pdf](https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf)
20. Chen Zheng, Lei Wang, Sally A. McKee, Lixin Zhang, Hainan Ye and Jianfeng Zhan “XOS: An Application-Defined Operating System for Datacenter Computing”, IEEE International Conference on Big Data (Big Data), 2018.
21. ONOS, “Architecture and Internals Guide - System Overview” [Online: accessed 06-2019] <https://wiki.onosproject.org/display/ONOS/System+Components>
22. ONF, “Our Mission” [Online: accessed 06-2019]  
<https://www.opennetworking.org/mission>
23. FiberMountain, “Is SDN the Ethernet of the 80’s or ATM of the 90’s?” [Online: accessed 06-2019]  
<https://blog.fibermountain.com/blog/is-sdn-the-ethernet-of-the-80s-or-atm-of-the-90s>
24. Open Networking Foundation “Software-Defined Networking: The New Norm for Networks” ONF White Paper, April 13, 2012.
25. SearchStorage, “Storage Hypervisor ” [Online: accessed 06-2019]  
<https://searchstorage.techtarget.com/definition/storage-hypervisor>



26. Gladys Diaz, Noemie Simoni “Network Service Description for Virtual Network Deployment: A constraints based OVF extension proposal”, 12th International Conference on Network and Service Management (CNSM), 2016.
27. Eurecom, Open Air Interface Alliance. Open-Air Interface Project [Online: accessed 06-2019]  
<https://gitlab.eurecom.fr/oai/openairinterface5g/wikis/HowToConnectCOTSUEwithOAIeNBNew>
28. Eurecom, Open Air Interface Alliance. Open-Air Interface Project [Online: accessed 06-2019] <https://gitlab.eurecom.fr/oai/openairinterface5g>
29. Eurecom, Open Air Interface Alliance. Open-Air Interface Project, “Basic Deployment of vEPC” [Online: accessed 06-2019]  
<https://github.com/OPENAIRINTERFACE/openair-cn/wiki/Basic-Deployment-of-vEPC>
30. Network Convergence Lab - Jeju National University, “Open Air EPC with NSSF Modifications” [Online: accessed 06-2019] <https://github.com/ncl427/openair-cn>
31. Open Networking Foundation, “CORD: Central Office Re-architected as a Datacenter” [Online: accessed 06-2019]  
<http://opencord.org/wp-content/uploads/2016/10/BBWF-CORD.pdf>
32. Open Cord, “CORD: Central Office Re-architected as a Datacenter” [Online: accessed 06-2019]  
<https://xosproject.org/wp-content/uploads/2018/08/CORD-XOS-Platform.pdf>
33. ONOS, “vRouter” [Online: accessed 06-2019]  
<https://wiki.onosproject.org/display/ONOS/vRouter>
34. ONOS, “CORD VTN” [Online: accessed 06-2019]  
<https://wiki.onosproject.org/display/ONOS/CORD+VTN>
35. Kapil Kumar, Nehal J. Wani and Suresh Purini “Dynamic Memory and Core Scaling in Virtual Machines“, IEEE 8th International Conference on Cloud Computing, 2015.
36. Molka Rekik, Khouloud Boukadi, Hanene Ben-Abdallah “A Context Based Scheduling Approach for Adaptive Business Process in the Cloud”, IEEE 7th International Conference on Cloud Computing, 2014.
37. Paolo Bellavista, Luca Foschini, Riccardo Venanzi, Giuseppe Carella “Extensible Orchestration of Elastic IP Multimedia Subsystem as a Service using Open Baton”, 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2017.
38. Keisuke Kuroki, Masaki Fukushima and Michiaki Hayashi “Framework of Network

- Service Orchestrator for Responsive Service Lifecycle Management”, IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015.
39. In-Yong Jung, Chang-Sung Jeong “Selective Task Scheduling for Time-Targeted Workflow Execution on Cloud”, 2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS), 2014.
40. Subhash Saini, Haoqiang Jin, Robert Hood, David Barker, Piyush Mehrotra and Rupak Biswas “The impact of hyper-threading on processor resource utilization in production applications”, 18th International Conference on High Performance Computing, HiPC 2011, Bengaluru, India, December 18-21, 2011.
-