Autoscaling Mechanism based on Execution-times for VNFM in NFV Platforms

Asif Mehmood*, Javier Diaz Rivera*, Talha Ahmed Khan*, Wang-Cheol Song*

Abstract

The process to determine the required number of resources depends on the factors being considered. Autoscaling is one such mechanism that uses a wide range of factors to decide and is a critical process in NFV. As the networks are being shifted onto the cloud after the invention of SDN, we require better resource managers in the future. To solve this problem, we propose a solution that allows the VNFM to autoscale the system resources depending on the factors such as overhead of hyperthreading, number of requests, execution-times for the virtual network functions. It is a known fact that the hyperthreaded virtual-cores are not fully capable of performing like the physical cores. Also, as there are different types of core having different frequencies so the process to calculate the number of cores needs to be measured accurately and precisely. The platform independency is achieved by proposing another solution in the form of a monitoring microservice, which communicates through APIs. Hence, by the use of our autoscaling application and a monitoring microservice, we enhance the resource provisioning process to meet the criteria of future networks.

Key Words: Autoscaling, Network Function Virtualization, Software Defined Networking, Clocks per Cycle (CPS)
1. Introduction

Innovation in networks has risen up to its peak after the invention of SDN [1]. It had a huge impact on networks. The price of the network devices got cheaper. Most of these network devices are general purpose devices as the control plane was separated from the data plane.

The network’s control plane after being centralized, forced the researchers to think about the development of these network function in a standard way. To solve the problem related to orchestration such as managing, normalizing the VNFs, autoscaling, monitoring and troubleshooting the resources etc, an architecture like NFV was proposed by standardizing bodies like ETSI [2] and OPNFV [3].

Even though, there are solutions to virtualize the resources but the performance of the networks has reduced. So, some of the solutions that exist now trade-off between the two factors of network performance and resource utilization.

The benefit of standardizing the VNF configuration or the model designed for it will allow them to be reused and will allow faster innovation in the field of telecommunication sector. It will also allow us to design and develop such systems which will be able to take decisions intelligently to dynamically scale the resources. So, overall the process of standardizing the VNF models will have a huge and positive impact on the resource provisioning area of Network Function Virtualization.

Our Intention was to develop a mechanism which could take the factors such as clocks per cycle [4] to decide the number of the cores to be assigned. It also used execution-times [5] for the requests received by an instance of a VNF.

Another factor that we used in our system was to utilize the weight-factor [6] to estimate the accurate number of cores for an instance of a VNF.

We used our own developed NSSF [7] in this system, which is integrated with the OAISIM [8] access-network and core-network services. We decided to select M-CORD [9] as a framework for the deployment and orchestration of the resources. M-CORD being the re-architected data center, provides us a full fledge framework with its own unique architecture which allows us to develop and run everything as a service. This property of running everything as a service is achieved by the XOS (Service Orchestrator). This module then utilizes the integrated Cloud and SDN projects within its domain to provision, orchestrate and manage the network resources. So, we use a platform which is not fully compliant to NFV standards but at least lets us design, develop and test our autoscaling mechanism for multiple type of NFV architectural systems.

Section II provides the necessary details related to literature. It also highlights the motivation of our work. Section III contains our proposed system design and then puts some light on its uniqueness. Related to our system, different attributes, benefits and drawbacks are discussed in detail. The benefits include the efficiency and platform independency architectures because of its API structure, used for communication among the components of our system. The factors involved in the decision mechanism of autoscaling are discussed in detail. Section IV evaluates our system in terms of CPU utilization. We compare our system results to show the efficient usage of the resources. In section V, we conclude the paper and propose the future work that needs to be done, for this work to be more worthy.
Ⅱ. Motivation and Related Work

As the purpose of future mobile networks is to provide seamless services and that is only possible if the services of the system are able to dynamically scale up or down. The development of architectures for virtual network functions has become a trend. Although, many researcher groups are proposing and developing this type of architectures to propose different solutions in the aspect of orchestration [10]. The organizations and researcher groups like ETSI, OPNFV, ONF [11] etc. are the dominant ones to propose solutions that aim to provide network slicing mechanisms for 5G networks. Using the mechanisms and frameworks based on the standards provided by these standardizing bodies, we can autoscale [12] the resources among each of the network slice depending on the usage of each network slice.

In the paper [12], author proposes an algorithm to dynamically allocate the processing capabilities and the memory to the virtual resources such as VMs. The main mechanism was to use the execution-times given for the processes that run inside the VMs. Then based on this real-time data they proposed a solution to assign resources based on other factors as well. This allows us to get the idea of using execution-times for the actions that the VNF can perform, leading us to use it along with the other factors too.

In the following paper [6], proposed system reduces the power consumption of cloud machines by allocating the cores and memory very efficiently. As it is known that when the resources are over-utilized, the systems heat up. Factors for the resource management of cloud computing services are carefully chosen in this paper, which yield better results in the form of optimal resource usage.

In the paper [13] related to data plane VNF’s autoscaling, author shows an abstract way to define the modular and necessary details to fulfil the criteria of autoscaling. The author used the network services used in the LTE Advanced architecture to propose network slicing.

The paper [14], in which author proposes the solution of using a custom monitoring drive to fetch the real-time data for the network services. This idea helped us in a way that using this approach we could reduce the latency caused by the cloud services deployed across different nodes. One more reason was that the custom monitoring application will reduce the extra latency caused by the communication done between intra cloud services.

One more paper [15], which is our previous work proposes a solution to create network slices based on contracts. This process provides a proof of concept and a development strategy to create network slices by using the layer on top of an orchestrator. This solution provides a way to automatically configure and provision the network slice and assign the necessary number of resources without the need of going into further details.

In this paper [5], the author proposes an algorithm to estimate the processing-time i.e. the execution-time for cloud application. We plan to improve our system to estimate the execution-times for the network services in the future. As the nature of networks is very diverse, so we feel the necessity to improve this area and it is one of the goals of NFV platforms. We can provide a high level of abstraction where multiple project vendors could deploy their services, whether they be open source or private source project vendors.

Ⅲ. System Design

We propose the following system shown in Fig. 1. It has two contributions in the form of applications and apart from it, modifications are also proposed in the management layer’s synchronizers [9].

The first contribution is the autoscaling application shown at the application layer of our architecture. This application includes modules such as Autoscale-controller, Information-handler and the Configuration-invoker.

Autoscaling-controller is the brain of our idea which takes the decision to scale up/down the resources for a VNF. This decision of autoscaling
is performed after a specific interval of time.

We denote this time by 't', which is the interval of an autoscale application. The detailed mechanism for the decision making is shown in Fig. 2, which is explained in the coming paragraphs. The autoscaling decision is based on factors including the clocks per cycle, execution-time, weight-factor etc.

Another module named as Configuration-invoker is responsible to generate the configurations for the underlying platform that is setup by XOS [16]. It receives these configurations in the form of TOSCA [9]. These configurations are either posted via a file or it could be posted via JSON APIs [17]. The point to be noted here is, that whenever the autoscale decision needs to be applied, Autoscale-controller requests the Configuration-invoker to generate and post the scale up/down configurations to XOS.

The Information Assembler which is responsible to store the necessary information of real-time data fetched by the Monitoring [14] microservice application. Autoscaling application works on the basis of heuristic statistics, with the help of Information Assembler. This store contains the number of requests received by the instance that is running in the domain of a specific VNFM. It also contains the execution-times for the actions/functions that a specific network-service or a VNF can perform. This approach of storing the execution-times for each action helps the Autoscale-controller determine the precise and accurate value of clocks per cycle required by a VNF instance for the next interval of time.
The above table also contains the VNF catalogue as well as the VNF-instances catalogue. VNFs must have 1-many relationship with the instances table. Also each instance must have at least one action/function. The requests table contains the information of a request for a specific action that was executed on a specific time-stamp. Every time a request is sent to the VNF instance, an entry/tuple is entered into this table which is stored by the microservice application.

Fig. 3 is the algorithmic representation for the mechanism we propose for Autoscale-controller. Firstly, we can see that it gets the list of VNFs prioritized by the order in which it needs to iterate the VNF instances. As we know that each VNF has a single or multiple instances. So for each instance it should iterate all instances for the VNF first and then to move onto the next instance. So first it gets the number of requests received by the instance and this information is provided by the Information-assembler. This algorithm estimates the cps ‘clocks per cycle’ required for this program to run. Each VNF has a threshold cps required. Now if the required cps is greater than the sum of minimum cps and threshold cps, it means that the instance needs to be scaled up.

1 Algorithm (Autoscaling Controller)

```
1: $PR = \{vnf_1, vnf_2, ..., vnf_f\}$
2: for all $vnf \in PR$ do
3:   for all instance of $vnf$ do
4:     initialize $cps_{fin} = null$
5:     query requests for instance instance
6:     calculate $cps_{req}$ for next interval
7:     setup the $cps_{min}$ and $cps_{max}$ for this instance
8:     if $cps_{req} > (cps_{min} + \psi_{thr})$ then
9:       $\psi_{core} = 1.0$
10:      if machine is virtual then
11:        $\psi_{core} = 0.8$
12:      end if
13:    end if
14: $cps_{fin} = \frac{\psi_{thres}}{\psi_{core}}$
15: end if
16: $GHz_{req} = cps_{fin} * 10^p$
17: end for
```

Fig. 3. Autoscaling decision (Algorithm)

The weight-factor is defined in order to measure or estimate the hyperthreading overhead caused by virtual-cores. In case of a virtual-core, it requires the weight-factor to be 0.8 which scales the final cps. Then the cps are converted into a GHz requirement and then according to the cores available in the system, our algorithm assigns the final number of cores to be assigned.

The above figure Fig. 4 shows the mechanism involved in the algorithm explained above. As we can see that the list of VNFs is iterated according to the priorities that we set for each of them. Then each instance of a VNF is iterated. This steps continues until all of the VNFs have been iterated for a specific VNFM.

Each iteration is divided into 7 steps. The first steps that it takes is to fetch the statistics for the actions that have been performed while functioning. Secondly, it calculates the cps from the execution-times and estimates the cps required for this program. The cores are estimated that are need for this program to run properly. Remember that there are other programs that need to run on the same virtual-resource (VM). It is not necessary that this is the only program that is running on the OS.

For the purpose to estimate the correct cps that other necessary system services or programs require which are supportive to the OS, we use cps threshold terminology and assign a specific threshold value for each of the VNF. Then we check if the resource needs to be scaled or not.

In th case of a virtual-core, we assign the weight-factor of 0.8 value so that we might not lose performance of the VNF. As this weight-factor value symbolizes the overhead for hyperthread. We divide the cps required with the weight-factor to increase the cps in case of a virtual-core as it can not perform equal to a physical-core. Finally, the cores are calculated by use of cps. As core calculation is done in GHz unit, so we may require 2 cores if one processor is of less frequency that the requirement.
IV. Evaluation and Results

The Fig. 5. shows the configurations, packages, softwares and tools required to setup the test environment.

Firstly, we use a physical system having the following specifications and then the deployment details are given in the second part of our table shown below in the form of a figure. It shows details such as database, language, IDE, hyperthreading capabilities etc. Table 1. shows the technical details as follows:

Table 1. Technical details for system environment

<table>
<thead>
<tr>
<th>Physical System</th>
<th>Deployed System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server Details</td>
<td>M-CORD</td>
</tr>
<tr>
<td>Server Hard Drive</td>
<td>3.9 TB</td>
</tr>
<tr>
<td>System Cores</td>
<td>16 (2.4 GHz)</td>
</tr>
<tr>
<td>Hyperthreads</td>
<td>2</td>
</tr>
<tr>
<td>Virtual Cores</td>
<td>16*(2) = 32</td>
</tr>
<tr>
<td>NIC</td>
<td>2</td>
</tr>
<tr>
<td>System Memory</td>
<td>64 GB</td>
</tr>
<tr>
<td>Operating System</td>
<td>Ubuntu 16.04.3 LTS</td>
</tr>
<tr>
<td>CORD Profile</td>
<td>M-CORD</td>
</tr>
<tr>
<td>Network Services</td>
<td>OpenAir Interface</td>
</tr>
<tr>
<td>M-CORD release</td>
<td>version 4.1</td>
</tr>
<tr>
<td>Nodes</td>
<td>corddev, head, compute</td>
</tr>
<tr>
<td>coredev CORES/RAM</td>
<td>1/2</td>
</tr>
<tr>
<td>head CORES/RAM</td>
<td>8/16</td>
</tr>
<tr>
<td>compute CORES/RAM</td>
<td>8/16</td>
</tr>
<tr>
<td>compute hyperthreads</td>
<td>5</td>
</tr>
<tr>
<td>compute virtual cores</td>
<td>8*(5) = 40</td>
</tr>
<tr>
<td>Database type</td>
<td>MySQL</td>
</tr>
<tr>
<td>Python plugin for db</td>
<td>SQLAlchemy</td>
</tr>
<tr>
<td>Languages</td>
<td>Python 3.6</td>
</tr>
<tr>
<td>Development IDE</td>
<td>PyCharm 2019.1.1</td>
</tr>
<tr>
<td>Hypervisor</td>
<td>kvm/libvirt</td>
</tr>
</tbody>
</table>
We use cpu utilization value to evaluate our proposed system. Table 2. contains the values of cpu utilization of all the assigned number of cpus per a VM/VNF. The values included are the average values accumulated for all of three cycles of tests. Each cycle of test had 3 user equipments configured to simulate the traffic to check the performance of our system.

The experimental statistics in a way represent that the number of cpu utilization is stabilizing itself up to 60%. As each of the time-interval represents that after each time-interval, the proposed autoscaling algorithm will run and then the cpu utilization comes to an optimal value, i.e. about 60 %. With the test case of 3 configured user equipments simulations, we ran three tests termed as a cycle. We accumulated those values and referred to them in a tabular form denoted as Table 2.

The results shown Fig. 6. show the stabilizing movement of cpu utilization caused by running our autoscaling algorithm after 7 intervals of time.

Each line represents a different VNF and somehow it is trying to increase the cpu utilization up to 60 %. As the usage is dependent on the requests received by the VNFs, indirectly saying that the number of traffic generated by the simulated UE. The system after running a number of autoscaling intervals gets to know the number of requirements to be fulfilled in order to stabilize the system and increase the resource usage up to optimal. In simple words, it tries to assign a number of resources based on the traffic expected.

To accumulate the statistics of cpu utilization for all of the VNFs together (instead of individual), we average and then round-off the values from the three cycles having 7 intervals each shown in Table 3.

The below shown Fig. 7. accumulates the statistics from all of the above cycle tests for all the VNFs together and depicts them as a graph. It means that, we are increasing the cpu utilization (for assigned cpus) of all the VNFs. This graph show the usage percentage of the cpus, which are in assigned (not the overall used/free cpus) to all the VNFs.
Now we present the statistics for the overall cpu assigned by our application. Both the statistics and the graph evaluate our results as a better performer as it can be seen that the number of resources are being freed up with the passage of time and then the system moves towards a consistent state ensuring the equal division of resources among the tenants efficiently.

### Table 4. Average usage of total cpu used

<table>
<thead>
<tr>
<th>Time</th>
<th>Usage % at time interval 't' for total CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
</tr>
<tr>
<td>t=1</td>
<td>32</td>
</tr>
<tr>
<td>t=2</td>
<td>32</td>
</tr>
<tr>
<td>t=3</td>
<td>32</td>
</tr>
<tr>
<td>t=4</td>
<td>32</td>
</tr>
<tr>
<td>t=5</td>
<td>32</td>
</tr>
<tr>
<td>t=6</td>
<td>32</td>
</tr>
<tr>
<td>t=7</td>
<td>32</td>
</tr>
</tbody>
</table>

The below Fig. 8 in graphical form depicts the statistics shown in above table. It shows that the number of free cpus at the first time-interval was 0 as all were occupied by the VNFs. Running the autoscaling algorithm after each interval of time, it can be clearly seen that the number of free cpus is being increased which reflects the fact that our autoscaling algorithm maintains the equal distribution of resources to save energy and to leave the free cpus for other tenants that are utilizing the infrastructure.

![Fig. 8. Average usage, total cpu](image)

### V. Conclusion

The scaling up and down of resources is a very critical decision to be made. While keeping in mind the key factors involved to determine the number of resources, we managed to fulfil the efficient usage of resources via an autoscaling and a monitoring application. The number of resources assigned were used up to optimum and the issue of under-utilization of resources is solved. Along with this, the number of resources were freed up for other tenants using the same infrastructure.

We plan to enhance the measurement process of execution-times for all of the network services. We plan to add network factors as well into the autoscaling mechanism that we proposed and developed. Tuning up weight-factors to measure the hyperthreading overhead needs to be improved or tested in diverse environments of networks.

### References


NFV 플랫폼에서 VNFM의 실행 시간에 기반한 자동 자원 조정 메커니즘

(HPCC, CSS, ICESS), 2014.


Asif Mehmood

2015 B.S. degree in Computer Science - COMSATS Institute of Science and Technology , Chak Shahzad Islamabad, Pakistan.

2017 ~ 현재 제주대학교 컴퓨터공학 석사과정

<관심분야> SDN, Intent Based Networking, 5G.

Talha Ahmed Khan

2017 B.S. degree in Computer Science - FAST-National University of Computer and Emerging Sciences, Karachi, Pakistan

2017 ~ 현재 제주대학교 컴퓨터공학 석사과정

<관심분야> M-CORD, 5G, Network Slicing
Javier Diaz
2005 B.S. degree in Computer Systems Engineering – Instituto Tecnologico de Estudios Superiores de Monterrey, Monterrey, Mexico
2017 – 현재 제주대학교 컴퓨터공학과 석사과정
<관심분야> NFV, SDN, 5G, M-CORD

송 왕철 (Wang-Cheol Song)
1986 - B.S. degree in Food Engineering – Yonsei University, Seoul, South Korea
1989 – B.S. degree in Electronics – Yonsei University, Seoul, South Korea
1991 – M.S. degree in Electronics – Yonsei University, Seoul, South Korea
1995 – PhD in Electronics, Yonsei University, Seoul, South Korea
1996 – 현재, 전임강사/조교수/부교수/교수 Jeju University