NFV-Enabled Vertical Scalability for IoT Slices

Hafizhuddin Zul Fahmi
EECS International Graduate Program
National Yang Ming Chiao Thung University
Hsinchu, Taiwan
hafizhuddin4@gmail.com

Fuchun Joseph Lin
Department of Computer Science
College of Computer Science
National Yang Ming Chiao Thung University
Hsinchu, Taiwan
fjlin@nctu.edu.tw

***Abstract*—With network slicing, 5G is able to best support heterogeneous IoT applications each with a specially tailored virtual network, called IoT slice. NFV (Network Function Virtualization) is the key enabler to virtualize a 5G network into multiple IoT slices for different types of IoT applications. All 5G network slices are required to support scalability for their operations in order to deal with dynamic fluctuation of incoming IoT requests. This research focuses on designing and implementing an NFV-enabled vertical scalability system for IoT slices according to their CPU loading. Our system is tested using a traffic generator with three types of IoT services. Each type of IoT services has a different QoS requirement, and the system will forward it to a specific network slice according to its requirement. We will evaluate the performance of vertical scalability by comparing two systems: one system implements vertical scalability followed by horizontal scalability while the other system implements only horizontal scalability. Our aim is to demonstrate that the former involves lower CPU utilization and power consumption while still achieves compatible response time and throughput when compared to the latter.**

Keywords—Vertical Scalability, Network Slicing, 5G, NFV, IoT

I. Introduction

The advent of 5G has significantly increased the number of connected IoT devices, enabling a wide variety of applications with different QoS requirements. Furthermore, by introducing network slicing, 5G will be able to support every type of IoT application with a specially designed virtual network, called IoT slice. NFV (Network Function Virtualization) is the key technology that enables the virtualization of a 5G network into multiple IoT slices. Each IoT slice is required to support scalability for its operations in order to handle the dynamic fluctuation of incoming IoT requests. Scalability can be carried out in two ways, either in horizontal or vertical manner. Horizontal scalability is to scale out or scale in the number of VNF (Virtual Network Function) instances in a network slice while vertical scalability is to scale up or scale down the capacity of a VNF instance without increasing or decreasing its number [1].

This research focuses on designing and implementing NFV-enabled vertical scalability for IoT slices according to their CPU loading. We will evaluate the performance of vertical scalability by comparing two systems: one system implements vertical scalability followed by horizontal scalability while the other system implements only horizontal scalability. Note that both vertical scalability and horizontal scalability will be executed by the MANO (Management and Orchestration) framework defined by European Telecommunications Standards Institute (ETSI). Our aim is to demonstrate that the system implementing vertical scalability followed by horizontal scalability involves lower CPU utilization and power consumption while still achieves compatible response time and throughput when compared to the system implementing only horizontal scalability.

The rest of the paper is organized as follows: Section II introduces the background information of IoT/M2M plaftorms, Tacker, Openstack and NFV-enabled scalability for IoT slices. Section III presents system design of NFV-enabled vertical scalability. Section IV describes our system testing and evaluation. Finally, Section V presents the conclusion and future work of this research.

II. Background

Broadly speaking, scalability can be carried out in two ways, either horizontal or vertical scalability. However, in the past, most of scalability research only focused on applying horizontal scalability. For example, in [2] a master node to track the amount of input and the percentage of CPU loading on virtual machines and take actions of scale-out and scale-in in order to achieve high scalability for a VM based IoT/M2M was proposed. Then, in [3] we proposed a highly scalable system for IoT/M2M traffic based on OpenStack. Instead of using OpenStack-native scalability methods, we designed a master node and a load balancer to perform horizontal scalability.

In this research, we focus on applying NFV-enabled vertical scalability to IoT/M2M systems based on the status of CPU loading. The decision to explore vertical scalability in our research is because applying horizontal scalability immediately after the system is overloaded may not be necessary. Vertical scalability can be applied first to scale up or down the capability of a VNF before applying horizontal scalability. By not increasing or decreasing the number of VNF instances we can reduce the cost of scalability when dealing with the changing load of IoT systems [4]. Both CPU and memory could be scaled up and down with the aim of increasing or decreasing the system capability at a lower cost than immediately applying horizontal scalability as in our previous research [5].

During experiments, several kinds of open source software were utilized. The first is OM2M developed by LAAS-CNRS [6] that provides the IoT/M2M service platform to be run as a VNF in order to construct network slices. Next is OpenStack, an open-source cloud operating system that provides many cloud services [7] such as Nova for provisioning compute instances, Neutron for networking services, Swift and Cinder for providing object and block storage, Keystone for providing identity and authentication services, Heat for service orchestration and Horizon for web frontend services. Then is Tacker, an open source project under OpenStack that implements NFVO and VNFM based on the ETSI NFV MANO framework. It can manage multiple VIM such as OpenStack and Kubernetes. In this work, Tacker and OpenStack were utilized to support the creation and deletion of network slices [8].

III. ARCHITECTURE AND DESIGN OF NFV-ENABLED HORIZONTAL AND VERTICAL SCALABILITY

In this section, we explain the architecture of NFV-enabled scalability, its system design including vertical vs horizontal scalability workflow and its components including master node, traffic generator, traffic monitor and load balancer.

A. Architecture for NFV-enabled scalability

The NFV-framework developed by ETSI has four main blocks: Operation Support System/Business Support System (OSS/BSS), Virtual Network Functions (VNFs), Network Function Virtualization Infrastructure (NFVI) and NFV-Management and Orchestration (NFV-MANO) as depicted in Figure 1. In this research, we use Tacker as NFVO & VNFM and OpenStack as VIM. The NFV enables the implementation of system scalability in terms of network services and network functions. To support scalability, we also design a Master Node in NFVO to support and implement the horizontal and vertical scalability mechanisms. The Master Node is integrated with NFVO via Tacker APIs. Within Master Node, monitoring functions are designed to detect the loading of each VNF in the NFV system. Finally, OpenStack is used to provide functionalities of VIM and NFVI.

B. System Design

New designs are required to implement scalability. First, we need to design a new NSD in accordance with our goals of implementing both horizontal and vertical scalability in the NFV framework. MANO is in charge of activating an NS, which is made up of a set of VNFs. Virtual Links (VLs) link these VNFs, and VNF Forwarding Graphs (VNFFGs) define the topologies of the VNFs. In order to automate the scalability of network slices, we rely on NSD and VNFD. The request from OSS/BSS to NFVO to create an NS (or a network slice) would include an NSD identifying the deployment flavor to be instantiated. The NSD consists of VNFDs, VL Descriptors (VLDs), VNFFG Descriptors (VNFFGDs) and other NSDs.

1) Vertical vs Horizontal Scalability Workflow

The vertical and horizontal scalability workflow is shown in Figure 2. This workflow is driven by the incoming IoT traffic load, We have set the overload threshold of the system at 55% and the underload threshold at 10% [3]. Based on these thresholds, either scale-up/out or scale-down/in will be triggered. If the monitor detects the CPU loading of the VNF instance is more than the upper threshold, the system will check whether the current capacity of the VNF is still upgradable. If yes, the VNF will be scaled up. Otherwise, the system will check if there is still sufficient resource to create a new VNF instance. If yes, the system will proceed with VNF scale-out. On the other hand, if the monitor detects that the CPU loading of the VNF instance is less than the underload threshold, the system will check whether the VNF can be downgraded and do scale down. Otherwise, the system will check whether

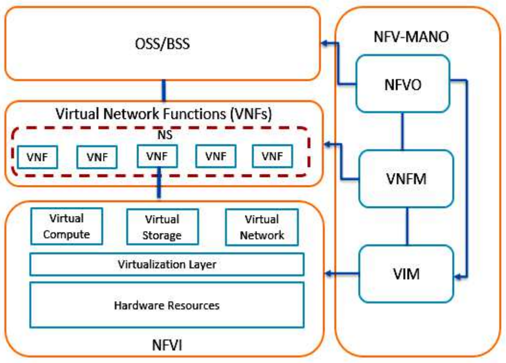

Figure 1. NFV Framework

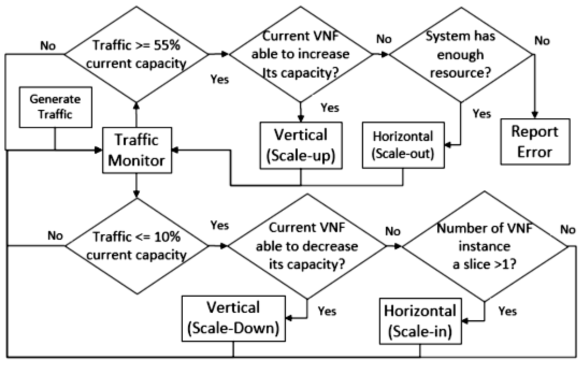
Traffic >= 55% current capacity
No
Yes
Current VNF able to increase its capacity?
No
System has enough resource?
No
Generate Traffic
Yes
Yes
Traffic Monitor
Vertical (Scale-up)
Horizontal (Scale-out)
Report Error
No
Traffic <= 10% current capacity
Yes
Current VNF able to decrease its capacity?
No
Number of VNF instance a slice >1?
No
Vertical (Scale-Down)
Yes
Horizontal (Scale-in)
Yes

Figure 2. Workflow for Vertical and Horizontal Scalability

the number of instances in the slice is more than 1. If so, the number of VNFs will be decreased (i.e. scaled in).

2) Master Node

Master Node contains both Traffic Monitor server and IL-Manager. It constantly checks the CPU loading of each OM2M VNF instance from Traffic Monitor client for underload and overload thresholds and trigger IL-Manager for scalability actions with the support of Tacker APIs.

3) Traffic Generator

We have designed a traffic generator to produce 3 different types of traffic with different payload sizes. The traffic generator will send traffic to the load balancer first. The load balancer then will distribute the traffic evenly according to the number of instances on the slice. These three types of traffic are used to emulate three different types of IoT applications including smart meter (low payload), smart parking (medium payload) and video surveillance (high payload).

4) Traffic Monitor

Traffic monitor in this research is divided into two separate parts: client and server. The client is located on an OM2M VNF instance that functions as a data collector to report the CPU loading and memory size of the instance. The server is located at the Master Node in NFVO, it receives data from the client and make scalability decision.

5) Load Balancer

We utilize HAProxy as a load balancer to split the traffic according to the predefined payload and subnet then send it to VNF instances on 3 different slices. The distribution or balancing of traffic between instances in each slice uses the roundrobin algorithm as a scheduler which will send each traffic to the OM2M instance in turn by RPC publisher [9].

IV. System Testing and Evaluation of Vertical vs. Horizontal Scalability

In this section, we describe our experimental environment, results and analysis when comparing our proposed system with a horizontal scalability system only system.

A. *Experimental Environment*

We use 2 physical servers with different capacities. Each is equipped with Ubuntu 18.04 OS; one with Intel E-52678V3 2.5 GHz processor and 128 GB RAM; the other with Intel 2.5 GHz 6 Cores processor and 64 GB RAM. Installed via DevStack on the first server are OpenStack with Stable Rocky version as a VIM and a traffic generator that we developed by ourselves. Then on the second server, we installed Tacker as NFVO and VNFM with Rocky version. Also on the second server, we have implemented the master node and the traffic monitor inside the NFVO. We use HAProxy as a Load Balancer and 2 OM2M VNFs as IoT platforms. The former is configured with 2 vCPUs, 2 GB RAM and 20 GB storage while the latter are configured with 2 and 3 vCPUs, 2 and 3 GB RAM and 20 and 30 GB storage for each of OM2M VNFs, respectively.

B. *Experimental Results and Analysis*

In this research, we compare the system implementing vertical scalability followed by horizontal scalability against the one implementing horizontal scalability only, in terms of CPU utilization, power consumption, throughput and response time. We evaluate the performances of both systems under three types of traffic with different payload sizes including a size of 380 bytes for smart meters, a size of 1000 bytes for smart parking and a size of 3000 bytes for surveillance video.

The traffic generator sends each traffic type to a different IoT slice during the testing, then the HAProxy on each slice distributes the traffic evenly to the OM2M platforms. All traffic types will be sent simultaneously to three different slices. There are three phases in our testing, in the first phase the traffic generator would send 10 requests per second within first 120 seconds. This phase would trigger the first scale up of the system. In the second phase, the traffic load would be doubled to 20 requests per second for 240 seconds, In this phase, the system will reach its peak load so that it can trigger both scale-up and scale-out. In the third phase, the system will run for 240 seconds by lowering the traffic load to 5 requests per second. This is done so that the system can scale in and scale down until it returns to the initial state.

The same three phases are also applied to the compared system, namely horizontal scalability only, with the same traffic load and total testing time for the purpose of comparison.

Figure 3 shows the evaluation results of CPU utilization between two systems. The system based on the horizontal scalability required higher CPU utilization than our proposed system in all three phases of testing. For example, the compared system obtained a value of 38.86% in the first phase, while our proposed system got 35.94%. Similarly in the second phase, it is 49.23% for the compared system and 47.50% for the proposed system. This trend continues into the third phase even with a decrease in CPU demands. As a result, we can conclude that the proposed system has the advantage of requiring less CPU resources than the compared system. This result occured because in the horizontal scalability only implementation, there are more than 2 OM2M/VMs running simultaneously all the time to handle the incoming traffic.

Figure 4 shows the evaluation results of power consumption by the server in watts between two systems. In evaluating power consumption we use the formula of Power Consumption $= TDP * CPU\% + K * Memory\%$ [10] to calculate the power consumption of each system where TDP represents the microprocessor's Thermal Design Power obtained from the product and for our physical server, it has a value of 85 watts; K is the general power consumption of memory modules, which is 6.258 W. Figure 4 shows that the compared system with horizontal scalability only consumes more power than our proposed system. For example, in the first phase the compared system consumes 35.4 watts of power while our proposed system only consumes 32.78 watts of power. Also in the second phase, our proposed system consumes power at 43.34 watts while the compared system consumes power at 44.92 watts. This trend continues until the system reaches its third phase. So it can be concluded that our proposed system consumes less power than the compared system due to the same reason as that for CPU utilization.

On the other hand, the comparison results of the throughputs for both systems are very close as shown in Figure 5. The throughput results of our proposed system are at 11.90, 13.79 and 12.95 requests/second that are very close to those of the compared system with horizontal scalability only at 12.18, 13.86 and 13.12 requests/second. The results show that the proposed system can achieve the similar level of throughput as that of the compared system due to timely actions of scale-up.

The comparison results of the response time are shown in Figure 6. Again, the response time results of our proposed

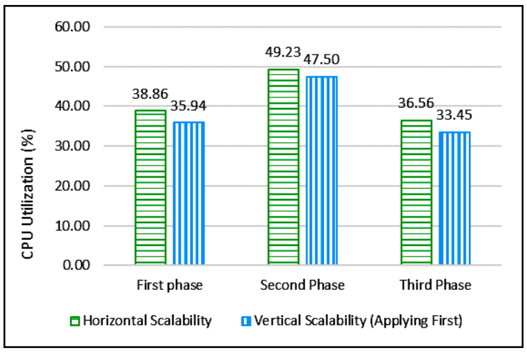

Figure 3. CPU Utilization Results with Three Slices System

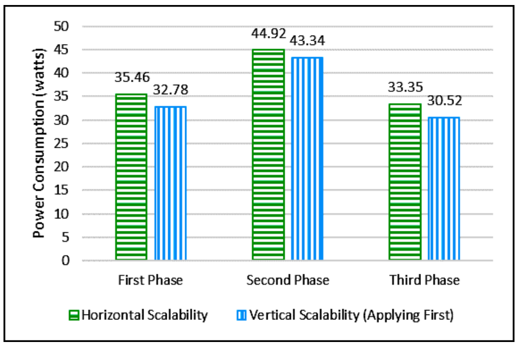

Figure 4. Power Consumption Results with Three Slices System

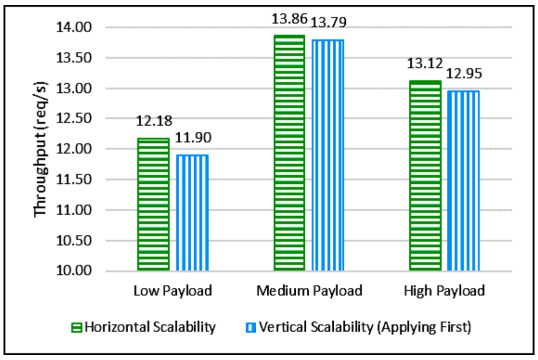

Figure 5. Throughput Results with Three Slices System

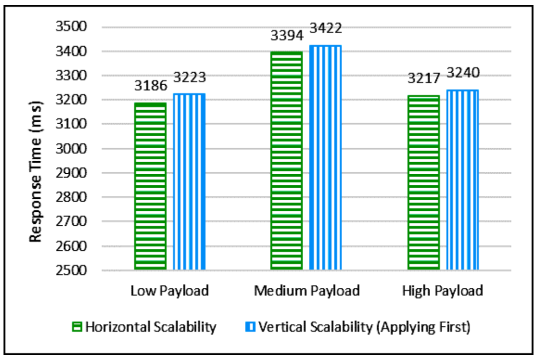

Figure 6. Response Time Results with Three Slices System

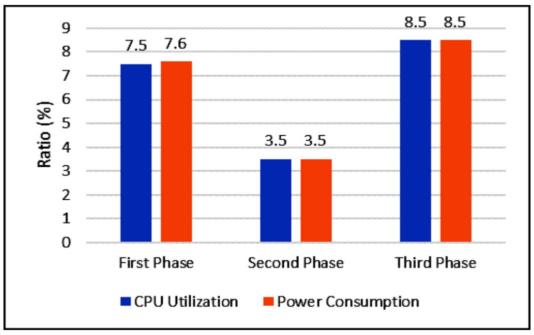

Figure 7. Ratio of Savings for CPU and Power Consumption between Compared System and Proposed System

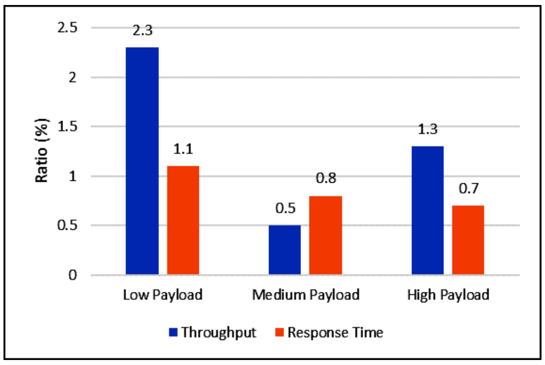

Figure 8. Ratio of Degradation for Throughput and Response Time between Compared System and Proposed System

system are at 3223, 3422 and 3240 ms that are very close to those of the compared system with horizontal scalability only at 3186, 3394 and 3217 ms. So, it can be concluded that the proposed system can achieve a similar level of response time as that of the compared system due to the same reason as that for throughput.

To make the comparison even clearer, Figure 7 and Figure 8 show the ratios of the differences between two systems in terms of both cost (i.e. CPU utilization and power consumption) and efficiency (i.e. throughput and response time) metrics. Figure 7 shows that CPU utilization and power consumption have the ratios of differences averaged around 6.5% for all testing phases, while in Figure 8, the ratios of differences of the two systems for throughput and response time are averaged around 0.9-1.4% for all testing phases. These two figures demonstrate that our proposed system is not only more cost-effective than the compared system but also keeps a similar level of efficiency with that of the latter.

V. Conclusion and Future Work

We used several open-source systems such as OpenStack, Tacker, HAProxy and OM2M to implement our proposed solutions. Our contribution is to propose vertical scalability and apply it first before horizontal scalability in order to achieve low cost utilization. The evaluation results show that our proposed system (applying vertical scalability first, then horizontal scalability) is more cost effective than applying horizontal scalability only while keeps close efficiency to the latter.

In the future work, it is worthwhile to explore the application of hybrid scalability to see whether we can achieve even higher scalability with the same environment. We can also explore the possibility of applying the ideas developed for IoT slices to 5G core network slices.

Acknowledgement

This work was supported by the Ministry of Science and Technology (MOST) of Taiwan under MOST 109-2221-E-009-083.

References

[1] A. Gupta, R. Christie, and R. Manjula, “Scalability in Internet of Things: Features, Techniques and Research Challenges,” *Int. J. Comput. Intell. Res.*, vol. 13, no. 7, pp. 1617–1627, 2017.

[2] F. J. Lin. and D. de la B. E. Cerritos, “High scalability for cloud-based IoT/M2M systems,” 2016 IEEE International Conference on Communication (ICC), pp. 1-6, May 2016.

[3] D. De La Bastida and F. J. Lin, “OpenStack-based highly scalable IoT/M2M platforms,” *Proc. - IEEE Int. Conf. Internet Things*, pp. 711–718, June 2017.

[4] M. A. Razzaq, “Smart campus system using internet of things: simulation and assessment of vertical scalability,” *Indian J. Sci. Technol.*, vol. 13, no. 28, pp. 2902–2910, 2020.

[5] T.Tsai and F. J. Lin, “Enabling IoT Network Slicing with Network Function Virtualization,” Journal of Advances in Internet of Things, Volume 10, Number 3, pp.17-35, July 2020.

[6] The Eclipse Foundation, “What is Eclipse OM2M?,” *2015*. Accessed on: April. 12, 2021 [Online] Available: https://www.eclipse.org/om2m/.

[7] Openstack, “Software.”. Accessed on: April. 15, 2021 [Online] Available: https://www.openstack.org/software/.

[8] Openstack, “Tacker - OpenStack NFV Orchestration.” Accessed on: April. 13, 2021 [Online]. Available: https://wiki.openstack.org/wiki/Tacker.

[9] Mitchell Anicas,” An Introduction to HAProxy and Load Balancing Concepts”. Accessed on: April. 15, 2021 [Online]. Available: https://www.digitalocean.com/community/tutorials/an-introduction-to-haproxy-and-load-balancing-concepts.

[10] F. J. Lin. and D. de la B. E. Cerritos, “High scalability for cloud-based IoT/M2M systems,” 2016 IEEE International Conference on Communication (ICC), pp. 1-6, May 2016.