The CSI Journal on
Computer Science and Engineering
Vol. 18, No. 1, 2020
Pages 1-12
Regular Research Paper

Multi-criteria reactive approach for joint dynamic VNF load balancing and service auto-scaling in NFV

Amir Kusedghi[1] Ahmad Akbari[1,2]

[1]School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran.
[2]Center of Excellence in Future Networks, Iran University of Science and Technology, Tehran, Iran

Abstract

The evolution of 5G key-enabler technologies, such as Software-defined Networking (SDN) and Network Function Virtualization (NFV), has brought network operators' attention to procure an efficient service delivery mechanism. Therefore, it is vital to leverage the service management and orchestration functionalities that operate in harmony. This can curb the undesirable negative impact of inconsistency on the quality of service. In this paper, we investigate the joint load balancing and auto-scaling of the elastic services that are being provisioned in the computing infrastructure at the edge or cloud. We address the necessity and challenges of designing the load balancing algorithm and scale decision making policy, aware of one another, through several practical scenarios on multimedia service delivery in NFV. It is demonstrated that changing the VNF load balancing method may deteriorate the scaling quality of decision-making. Consequently, we propose a novel multi-criteria reactive approach using a mutual score to dynamically adapt the VNF load balancing algorithm with the auto-scaling engine. We implement the proposed joint method on the management and orchestration layer of our previously developed NFV/SDN testbed, called XeniumNFV, to evaluate the effectiveness of our approach by conducting extensive experiments on the elastic web service.
Keywords: VNF load balancing, Service auto-scaling, NFV, SDN, Edge computing, 5G networks.

1. Introduction

The Fifth Generation (5G) network is a promising solution to meet the user requirements in terms of data rate, supporting mobility, improving capacity, delay, and so on. Traditionally, the performance of the networks heavily relies on the middleboxes that are deployed in the network and the input traffic must pass through them according to Service Function Chaining (SFC) concept. The advent of Software-Defined Networking (SDN) has brought the required flexibility in traffic steering by separating the control plane and the data plane providing more programmability with the global view of the network. On the other hand, Network Function Virtualization (NFV) prescribes decoupling of software components from their respective dedicated hardware to run on inexpensive commodity servers. In this way, virtual network functions can be instantiated on the NFV Point of Presence (NFV-PoP) in an agile and on demand manner which results in the reduction of the Capital and Operational Expenditure (CAPEX and OPEX).

In such NFV and SDN enabled 5G network, Virtual Network Functions (VNFs) can be created/removed rapidly on computing infrastructure in different locations, and the respective traffic would be steered efficiently towards them. The most important VNFs in this context can be categorized into groups providing or improving: network functionalities and connectivity (e.g., NAT, DNS, Load Balancer), network security (e.g., IDS, Firewall, DPI, Malware Detector), network performance (e.g., Traffic shaping, Rate Limiting, Accelerators), control functions and applications of telecommunication (e.g., EPC components, RAN functions), and elastic services (e.g., IMS, VoIP, web services). The first three types are typically used as a predefined sequence in the network forming a chain of VNFs called Service Chain (SC). The fourth group includes control functionalities that the data

plane traffic does not necessarily need to traverse them. The last group contains the traditional elastic applications that the customer's traffic must be distributed among a variable number of their instances.

Telco and service providers must adopt an elastic approach of service delivery in order to be responsive to the increasing amount of service requests while considering incurred expenses in terms of resource management. For this reason, all the aforementioned VNF types may need to be dynamically scaled (horizontally/vertically) in real time to cope with the traffic demands. As auto-scaling is considered as the prevalent solution to provide service elasticity and overload control, load balancing is a substantial and complementary functionality that can be enforced by either a load balancer VNF or SDN controller. Although auto-scaling and load balancing have been studied and used in many previous works, there is still a gap between individual solutions and a joint practical approach in the NFV context.

In this paper, we investigate the joint load balancing and auto-scaling of service instances that are being provisioned on the computing infrastructure at the edge or cloud. We elaborate on the necessity of designing the load balancing algorithm and scale decision making policy, aware of one another, through several practical scenarios. First, we show that choosing the load balancing and scaling policy separately would cause inefficient scaling decisions. Consequently, we propose a reactive approach that considers different criteria to calculate a score for each service instance that is used in both scaling policy and load balancing algorithm. We adopt a reactive approach as the scaling should be triggered in real time to coordinate with the load balancing preventing service degradation and overload/underload situation. However, the proactive approach could contribute to the scaling process as complementary to the reactive solution by predicting either the incoming traffic or the state of the service key performance indicators. Even though anticipating the scaling decision parameters ahead of time could be beneficial in terms of service availability, it needs a comprehensive study as an extension to this work. The key contributions of this paper are summarized as follows:

- *Assessment of auto-scaling policy and load balancing algorithm*: To demonstrate the significance of the auto scaling policy and load balancing algorithm, we conduct several real environment scenarios on elastic multimedia signaling service. We deploy the load balancer and signaling servers as containerized VNFs. We show that different scaling policies trigger auto-scaling at different states which may not be desirable. Moreover, we show that adopting different load balancing algorithms may affect the Quality of Service (QoS) and resource status of the VNFs regardless of the scaling policy.
- *Joint VNF load balancing and service auto scaling*: Choosing a good scaling policy and an effective load balancing algorithm unaware of one another does not necessarily result in high quality decision makings. Therefore, we evaluate the effectiveness of a joint VNF load balancing and service auto scaling with multimedia use case through the proposed weight matching of the load balancing and scaling for a specific scenario. In this paper, we utilize the orchestration and monitoring framework of XeniumNFV [1] to run a real case study of elastic SIP servers in the SDN/NFV environment. We also discuss the required interactions between NFV units to fully implement this stateful application.
- *Multi-criteria reactive approach*: To provide the required awareness in a generic solution of the joint VNF load balancing and service auto scaling, we propose a novel multi-criteria reactive approach in which the scaling and load balancing decision makers use a common score function in their threshold-based, majority-voting scaling policy and dynamic weighted load balancing algorithm. We evaluate the proposed method using an elastic web service use case with a well-known load balancer VNF under several realistic traffic patterns.

The remainder of this paper is organized as follows. Section 2 introduces some related works and describes the implementation architecture for the proposed method. In Section 3, we present the contributing factors to system efficiency for joint VNF load balancing and service auto-scaling through comprehensive real environment experiments. According to the achieved results, we propose a multi-criteria reactive approach for joint dynamic VNF load balancing and auto-scaling in Section 4. In section 5, we evaluate the effectiveness of our proposed approach by conducting extensive experiments on the elastic web service. We conclude the paper and introduce possible future works in section 6.

2. Related Work and Background

Traditionally, server load balancing techniques are considered as resource management technique that can improve the QoS in the network. There are numerous well-known methods of load balancing that can be generally categorized as static and dynamic. Static methods use a predefined policy for load balancing decisions while dynamic algorithms adapt their policy due to real-time status of the network. In [2], the authors propose a dynamic weighted load balancer for web servers in the cloud considering the weights as decision making factors. Authors in [3] use load and channel agents for dynamic load balancing. The work in [4] proposes a weighted SLA-aware load balancing with a load prediction module that mainly takes the resource utilizations into account. There are also several researches that study the load balancing only for specific applications such as [5], which targets SIP servers. Similarly, [6] proposes weighted round robin load balancing for vMMEs in NFV-based Evolved Packet Core (EPC) changing the weights adaptively based on the latency of the links and the number of allocated CPUs. There are also some efforts that do not only balance the load between VNFs, but also try to provide network load balancing [7]. All these approaches consider a fixed number of servers or VNFs which is not a valid assumption in current and future networks.

Another way to classify the previous work is to consider the logical location where the load balancing algorithm is developed. Some older researches place the load balancer physically at the entrance of the network in a fixed location while others take advantage of two emerging technologies (i.e., SDN and NFV) to deploy the load balancer. The first

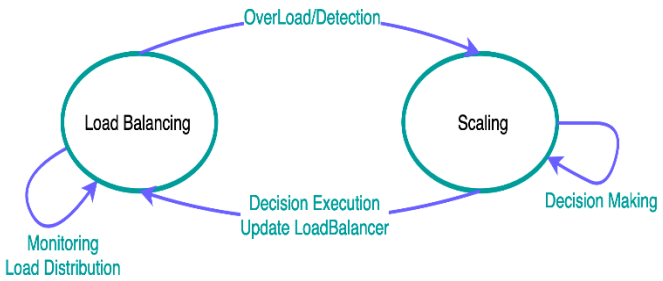


Figure 1: Finite state automata model for LB and scaling problem.

group deploys the algorithm on the SDN controller [8] in the control plane, and the second one instantiates the load balancer as a VNF in the NFV environment [9]. For instance, the approach in [10] implements the load balancing algorithm as a VNF and the SDN controller permits this VNF to redistribute the traffic among potential destinations. In this paper, we implement the load balancer as a VNF for two reasons. First, there are numerous types of elastic services being delivered in the 5G network that each one may require its own application specific algorithm. Therefore, developing a load balancing module on the SDN controller for each service type would cause tremendous overhead and the SDN controller would become the performance bottleneck of the network. Second, we can create the load balancer VNFs dynamically, any time and wherever needed, using containerized NFV. The NFV Management and Orchestration (MANO) would be responsible to notify the SDN controller about the location of each load balancer VNF.

In addition to load balancing, auto-scaling is a promising approach to cope with the fluctuation of the traffic demand without degradation of the QoS. But since increasing the number of instances associated with an elastic service imposes an extra cost, we should avoid unnecessary scale-outs. In this paper, we only consider horizontal scaling due to limitations of vertical scaling in real world scenarios mainly caused by distributed nature of computing resources. By taking advantage of the container-based VNFs, the overhead of creating a new instance is considerably reduced, and as explained in [11] the scaling performance is much better compared to virtualization. However, [12] devises a hybrid solution for container-based services leveraging high availability of the horizontal scaling and the fine-grained resource control of the vertical scaling. Authors in [13] aim to tackle the same problem using Reinforcement Learning (RL) with more flexibility. Nonetheless, debating about the trade-off between horizontal and vertical scaling is out of our scope in this paper.

Existing approaches on VNF auto-scaling mostly focus on two main aspects. The first group tries to select proper metrics for scale decision making while the other group propounds solutions to predict the changing pattern of these metrics or forecast the traffic itself in order to proactively scale the service ahead of time avoiding overload situation. The survey in [14], studies auto-scaling of the web applications in the cloud and our proposed method can perfectly fit in with the attributes of the explained taxonomy, but our context is specifically containerized SDN/NFV. In [15], the authors aim to predict the resource requirements of different components in IP multimedia Systems (IMS) by monitoring and learning

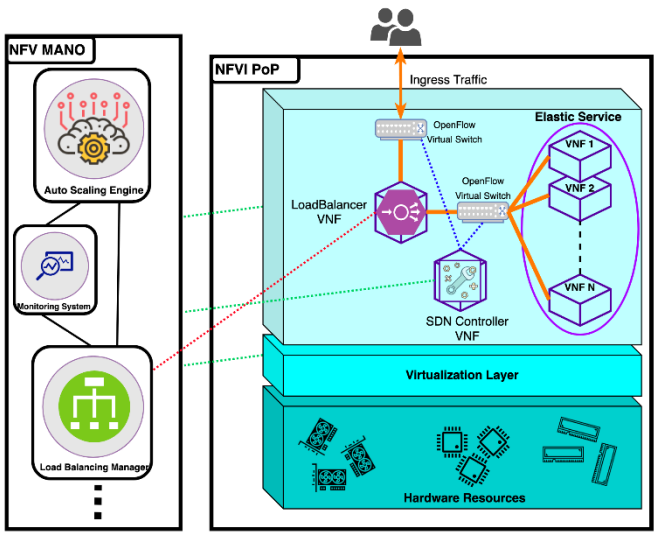


Figure 2: Logical architecture of joint VNF load balancing and service auto-scaling.

the states of neighbor components that are directly linked to them. z-Torch [16] is another NFV orchestration and monitoring solution that aims to improve the quality of scale decision making through analyzing the key performance indicators and their affinity. There are also papers devising solutions for scaling of the service chains, for example [17] proposes a scaling scheme for geo-distributed service chains in mobile core networks and IMS. An extensible framework for elastic service chains in 5G networks is introduced in [18] using open source projects for life cycle management and failure recovery. Authors in [19] and [20], adopt a proactive approach to predict the incoming workload and the respective response time of the system to make scaling decisions for distributed services and multi-tier applications. Similarly, [21] proposes a load forecasting solution for the 5G mobile core in order to trigger incremental scaling of the AMF component. These works ignore the significant impact of the LB algorithm and scaling policy on one another as they only make a decision based on the system performance.

Nevertheless, VNF load balancing and auto-scaling are considered as the main solutions for overload control in NFV and 5G networks. Authors in [22], use traffic throttling in three layers of VNF, host, and network as a complementary method for overload control. Another recent work [23] suggests that in the case of CPU contention the regular throttling may not suffice. Although scaling can resolve the problem in general, during traffic spikes it does not work well. We believe, if load balancing and scaling solutions work aware of each other, they can effectively avoid over/under load. In this way, we can consider them as states of a finite automaton for overload control of elastic services that is depicted in Fig. 1. Furthermore, we adopt a reactive approach to balance the incoming load among existing service instances to hinder unnecessary scale-outs. However, upon overload detection, we add another service instance and update the load balancing algorithm dynamically to cope with the scaling decision.

Even though some of the previous studies consider the VNF load balancing and scaling, but they mostly apply them completely independent of one another. As mentioned before, proactive approaches mostly ignore how load balancing is deployed in the network, but even hybrid solutions of proactive and reactive such as [24] and [25], assume a

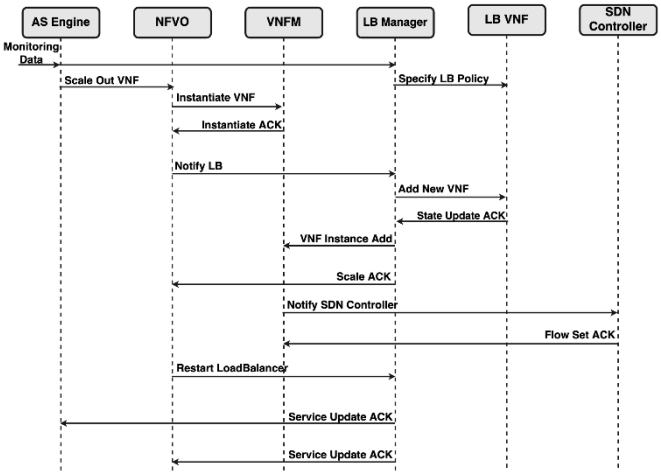


Figure 3: Sequence diagram of scale out.

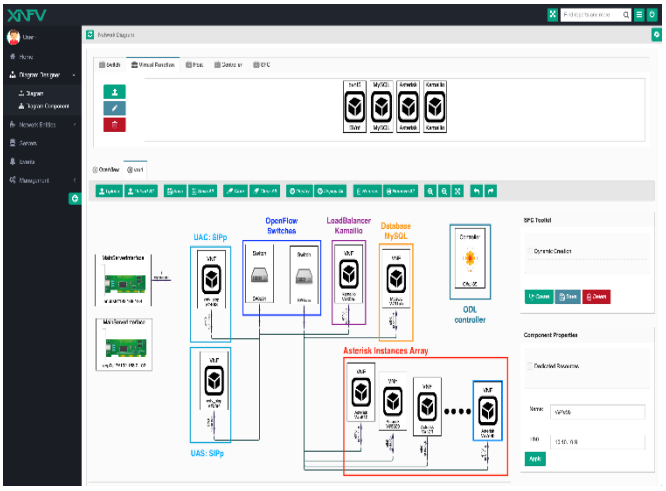


Figure 4: Implemented SIP scenario in XNFV.

balanced state or deploy a simple static load balancing between VNFs. Benifa *et al.* [26] employ a Continuous Time Markov Model (CTMM) for load balancing in their hybrid solution, but the scaling and load balancing do not coordinate with each other. In another work [27], the authors propose a novel architecture to support autonomic service management using NFV/SDN, monitoring system, and demand prediction. Although VNF auto-scaling is provisioned for the CDN use case, it does not investigate the efficiency of the load balancing method and scaling policy together. Similarly, [28] embeds both these functionalities in the network and their compute-aware orchestration of multimedia services mostly focus on the load balancing between SIP servers. Authors in [29] deploy the load balancer in the SDN controller and consider the auto-scaling of web service. The load balancing algorithm distributes the traffic among service instances based on the number of flows assigned to them during the last decision window, and the scale out/in decisions are made due to a threshold-based score function of resource utilization. However, even though the auto-scaler interacts with the load balancer to add the new instances to the pool, the load balancing and scaling policies are working independently and unaware of how the other one makes decisions.

The logical components involved in a joint VNF load balancing and auto-scaling of the SDN/NFV environment are shown in Fig. 2. We deploy the load balancing manager and auto-scaling engine as MANO units. The auto-scaling engine informs the load balancing manager about scale decisions and the manager updates the load balancer configuration and SDN controller accordingly. To elaborate on how interactions are made throughout this structure, we present the sequence diagram (Fig. 3) for the implementation of a scale out decision in this framework.

Another new paradigm of service orchestration uses VNF load balancing and auto-scaling solutions to increase the availability and reliability of 5G networks. Availability and reliability, alongside failure detection and recovery, are considered as dependability attributes. In [30], the availability of telecommunication services is preserved using auto-scaling to dimension the bottlenecks of the virtualized mobile core (e.g., EPC and IMS). In this context, the dependability of the NFV orchestrator is comprehensively discussed in [31]. The VNF placement and resource allocation is another main task of NFV orchestration that contributes to service quality and dependability which is discussed in [32].

3. Motivation and Challenges

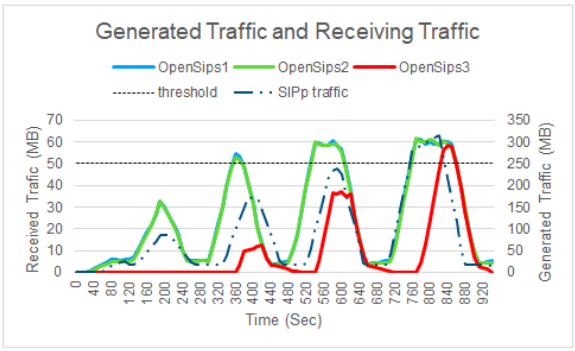
The main purpose of service auto-scaling is to meet the desired service level agreement (SLA) considering the resource allocation cost in case of traffic demand fluctuations. Allocating more resources is only tolerable if it prevents SLA violations. On the other hand, load balancing between service instances is a propitious solution to make use of the resources efficiently resulting in a smaller number of scales and incurred cost, accordingly. This section gives a good insight into the problem and identifies the key factors affecting the decision of adding/removing instances. The previous works have addressed the scaling and load balancing problems separately. In this paper, we adopt a joint approach for determining the scaling policy and load balancing algorithm. It should be noted that the feasibility of this idea is tightly coupled to virtualization and softwarization. We believe that the following three factors significantly contribute to the system performance and they should be considered precisely to guarantee the desired QoS and cost efficiency.

Scaling policy: The first factor affecting the number of VNF instances to maintain service quality is the scaling policy. The number of instances for a particular service may differ when the scaling policy is based on resource utilization (e.g., CPU, RAM), traffic rate, etc.

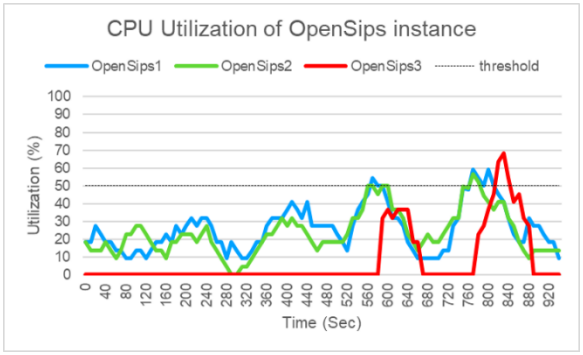
LB algorithm: We believe that the load balancing algorithm has a significant effect on the number of required service instances. It is possible that adding additional service instances does not improve the QoS due to the inappropriate load balancing method.

Awareness between scaling policy and LB algorithm: Another substantial factor is the harmony between the scaling policy and the LB algorithm. This means that not only the LB algorithm affects the scaling problem, but also matching the LB and the scaling policy plays an important role that must be precisely considered.

In this section, we conduct extensive experiments to validate each of the above-mentioned points.



(a) Generated traffic by SIPp - Received traffic at servers.



(b) CPU utilization of SIP servers.

Figure 5: Scaling with different policies in experiment 1.

3.1. Proof of concept environment

To illustrate the importance of the aforementioned factors, elastic SIP application is used as a target service to investigate the joint LB and scaling problem. Fig. 4 is an image of the XeniumNFV environment in which the designed scenarios are implemented. SIPp [33] is used as UAC/UAS to send/receive the traffic in the SIP scenario. The SIPp traffic generator is configured to increase the call rate by 20 cps every 20 seconds until the maximum limit of 300 calls is reached. Asterisk [34] and OpenSIPS [35] are used as SIP servers in the experiments. The Kamailio load balancer [36] is used to distribute the traffic load between the SIP servers. All of these components were containerized and put together in the XNFV framework. XNFV is responsible for supervising the containers and performing the described scaling policy that is designed as an event. Table 1 lists the LB algorithms, scaling policies, and allocated resources to SIP servers for the following three sets of experiments.

3.2. Impact of scaling policy

Choosing a suitable scaling policy is critical in determining the number of VNFs. According to Table 1, two OpenSIPS servers with the same specification are created. The Round-Robin (RR) algorithm is chosen for load balancing to show the effect of scaling policy on the number of scale attempts. A new instance is created (or deleted) as soon as the receiving traffic in the SIP server exceeds (or falls short of) 50 MB. The generated load by UAC, the received traffic by UAS and the scale out times based on the first scaling policy are depicted in Fig. 5 (a). In the other test, the scaling policy is based on CPU utilization. The Scale out is only triggered when the CPU utilization reaches 50%. Fig. 5 (b) shows that in this case, the scaling operation is performed twice which is less than the previous test.

3.3. Impact of LB on auto-scaling

The second set of experiments aims to study the impact of two well-known and simple LB algorithms, Round-Robin (RR) and Weighted Based Distribution (WBD) on the system. The RR algorithm is selected as an example of a set of LB algorithms that do not consider resource capacity. The RR load balancing is a simple way to distribute requests across a group of instances in turn and finally go back to the top of the list and repeat again. The WBD algorithm is chosen as an LB algorithm that is aware of resource capacities and chooses the appropriate weights for each instance statically. The weight of each instance is determined according to equation 1.

Table 1: Specification of SIP tests.

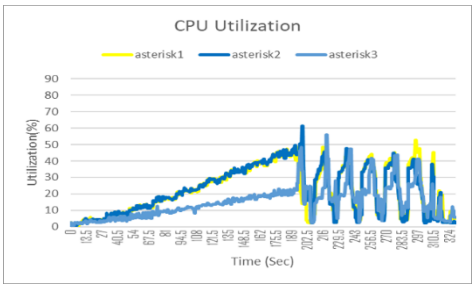
Effective factor	Scaling policy	LB algorithm	SIP servers
Scaling policy	RR	1.CPU utilization 2.Rx traffic	OpenSIPS1: 1vCpu,500MB RAM OpenSIPS2: 1vCpu,500MB RAM
LB algorithm	1.RR 2.WBD (W~resource)	α*Ucpu+β*Umem > threshold	Asterisk1: 1vCPU,400MB RAM Asterisk2: 1vCPU,400MB RAM Asterisk3: 0.5vCPU,200MB RAM
Awareness between LB and scaling	WBD (W~memory)	1.Memory utilization 2.CPU utilization	Asterisk1: 2vCPU,400MB RAM Asterisk2: 2vCPU,400MB RAM Asterisk3: 1vCPU,800MB RAM

$$W_i = \frac{Res_i}{\sum_{j=1}^{n} Res_j} \quad (1)$$

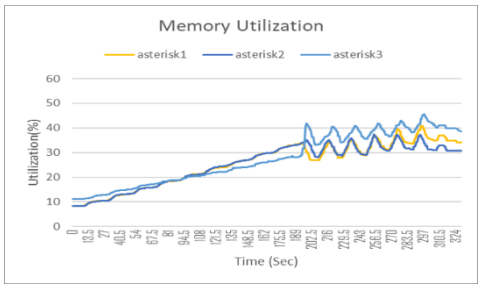
In this equation, Res_i is the amount of resource allocated to instance i, and W_i is the weight given to this instance in the LB algorithm. In WBD algorithm, more volume of traffic would be forwarded to a server with a larger weight. The scaling policy is also set with respect to CPU and memory utilizations. NFV management and orchestration monitors various parameters and automatically creates a new VNF only if the cost function of those parameters exceeds a specified threshold. Other approaches, such as fuzzy logic and time series analysis, are also applicable. To this end, this action is performed as follows:

$$\alpha * Ucpu + \beta * Umem > threshold \quad (2)$$

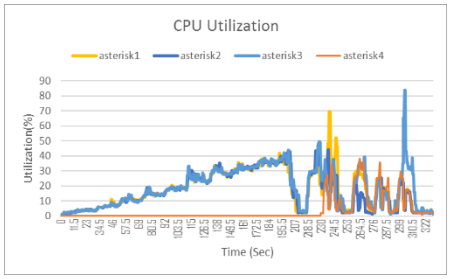
In the above equation, $Ucpu$ and $Umem$ are the CPU and memory utilizations of the VNF. Let the α and β be the coefficients that identify the relative importance of the resource utilization due to the application type.



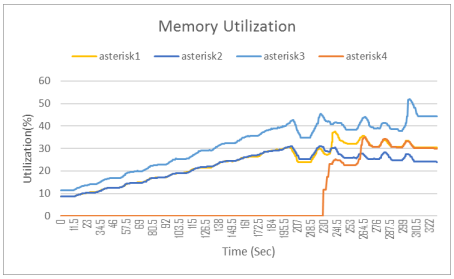
(a) CPU utilization for weighted LB.



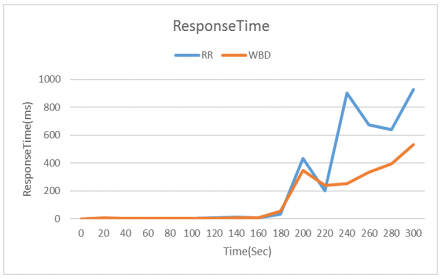
(b) Memory utilization for weighted LB.



(c) CPU utilization for round-robin LB.



(d) Memory utilization for round-robin LB.



(e) Impact of LB on response time.

Figure 6: Resource utilization and performance evaluation in experiment 2.

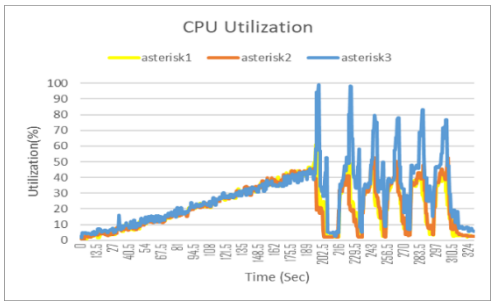
In order to illustrate the impact of load balancing on auto-scaling, the coefficients α and β, and the threshold value for scaling policy are set to 0.2, 0.8, and 45%, respectively. Fig.6 (a)-(d) shows the resource utilization of each SIP server when RR and WDB algorithms are deployed. Since the RR algorithm does not consider the capacity of SIP servers, the resource utilization of Asterisk3 increases constantly. Consequently, the new instance is created even though Asterisk1 and Asterisk2 are not yet fully utilized. In contrast, the weighted LB algorithm distributes the requests proportional to the resource capacity of each server. Consequently, the scaling process is never triggered and no new instance is created. This issue becomes even more critical when the performance parameters such as response time, deteriorate despite of adding a new instance. The response time of the service in each test is shown in Fig.6 (e). Since Asterisk3 has fewer resources than other instances it becomes saturated when the RR algorithm is deployed. subsequently, the response time is higher than running the experiment with WLB. Another notable point in this chart is that even though the response time has decreased slightly with the creation of the new instance, it still does not outperform the weighted LB simply because the RR algorithm does not distinguish between instances and sends the requests to Asterisk3 as much as others. Accordingly, the load balancing method has a significant impact on the auto-scaling performance. Moreover, it is evident that adding resources to the elastic service does not necessarily lead to better QoS.

3.4. Impact LB and auto-scaling awareness

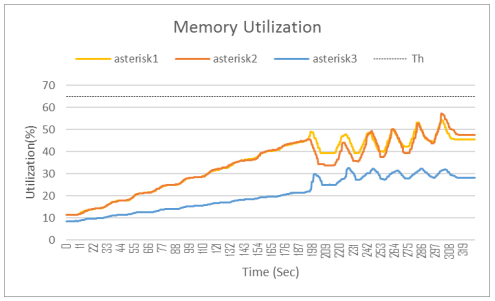
The third set of experiments elaborate that adopting a proper load balancing technique alongside auto-scaling is not necessarily effective if the load balancing algorithm and scaling policy do not match. Generally, the weighted LB algorithm outperforms the RR algorithm because of the resource capacity consideration that is thoroughly discussed earlier. In this section, we show that the inconsistency of load balancing and scaling policy degrades the performance of the system. We deploy the WBD load balancing algorithm for two tests. We set the weights of the SIP servers to 0.4, 0.4 and 0.2 (proportional to their CPU capacity). In the first test, the scaling policy is based on CPU utilization. A new instance is created if the CPU utilization exceeds 85%. Fig.7 (a) and (b) show the CPU and memory utilizations of all three SIP servers. The utilization of Asterisk3 exceeds the threshold first, due to its lower capacity. Consequently, a new instance is created and added to the instance pool. Therefore, the acceptable quality of service is preserved. In the second test, the scaling policy is based on memory utilization and a new instance is created if the memory utilization exceeds 65%. In this case, none of the SIP servers reach the threshold (Fig.7 (c)) even though Asterisk3 is overloaded and does not perform properly (Fig.7 (d)). In fact, due to incorrect matching of the scaling policy and load balancing, no new instance is created.

4. Proposed reactive approach

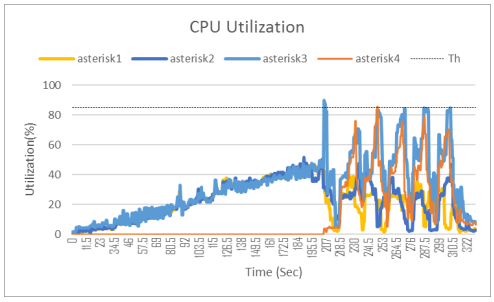
Given the determinative factors of an effective joint VNF load balancing and auto-scaling in the previous section, we propose a novel reactive approach for elastic services that are being delivered in NFV. Considering the fact that the key performance indicators of the different elastic services may vary from one application to another, we adopt a multi-criteria approach in order to provide a generic solution that only has to be customized based on the use case. The previous works and even commercial industry platforms mostly use resource utilization as their decision metric.



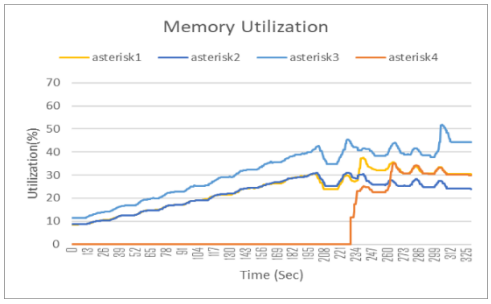
(a) CPU utilization with scaling based on memory.



(b) Memory utilization with scaling based on memory.



(c) CPU utilization with scaling based on CPU.



(d) Memory utilization with scaling based on CPU.

Figure 7: Resource utilizations in experiment 3.

In this paper, we consider three types of metrics for decision makings in auto-scaler and load balancer. Eq. 3 represents the resource utilization metric as a function of CPU and memory utilization. S_t is the set of current VNFs associated with the elastic service at time *t*. The coefficients must be tuned to cope with the resource consumption pattern of the application. For example, for a CPU-intensive application the coefficient of the CPU utilization should be more than the coefficient of the memory utilization.

$$\forall VNF_i \in S_t: U_{VNF_i} = (\alpha * U_{CPU_i} + \beta * U_{mem_i})/(\alpha + \beta) \quad (3)$$

The second category of metrics is performance. Regardless of the VNF's resource state, the current status of the performance characteristics is a good decision variable, especially when there are SLA terms that must be guaranteed. The third type includes the application specific parameters. The operational statistics of the application are proper metrics to demonstrate the quality of the functional behavior of the service. For instance, if the false positive of an intrusion detection system increases unexpectedly, this implies that either the processing power is not enough for the current load or the rules are not properly set. Let the U, P, and A denote the total resource, performance and application utilization, respectively. All these variables are normalized and each individual may be a weighted arithmetic mean of some parameters within their category that are calculated the same way as Eq. 3.

We define a *score* for each service instance as the weighted mean of the three metric types. The coefficients in Eq. 4 are set according to the importance of the metric tuned by the administration preference and the application nature. The *score* of each service instance, within each decision window, is calculated and shared with both scaling policy and load balancing algorithm. Fundamentally, having a common *score* as the decision metric to evaluate the status of the VNFs in both auto-scaling and load balancing contributes to provide awareness between these functionalities.

$$score_{VNF_i} = \frac{(\gamma * U_{VNF_i} + \delta * P_{VNF_i} + \varphi * A_{VNF_i})}{(\gamma + \delta + \varphi)} \quad (4)$$

To achieve an effective joint load balancing and auto-scaling solution, we adopt a reactive approach based on the multi-criteria score. Most of the existing works in the literature on reactive scaling policies acquire a threshold-based decision making. To tackle the gap between the load balancing effectiveness and auto-scaling turnover on the system performance, we propose a combination of threshold-based and fuzzy logic scale decision making. Moreover, we devise a dynamic load balancing algorithm in order to match the decisions of the load balancing and scaling in every iteration. In this way, we update the load balancing algorithm in an online manner. The proposed algorithm for the joint dynamic VNF load balancing and auto-scaling is depicted in *Algorithm1*.

The first few lines of our proposed algorithm (1-8) provide the decision metrics for both load balancing and auto-scaling policy. The flag *state* represents the current status of the service instances as *overloaded*, *underloaded*, and *normal*. We define an *upperThreshold* and *bottomThreshold* to set the state of each VNF. In addition to the state, the *weights* of our proposed dynamic weighted round-robin algorithm are calculated using each VNFs score. Since the score associated with each VNF represents the current efficiency of the corresponding VNF, it can be used to specify the weights of the load balancer. The more each service instance is occupied, the less traffic load balancer should redirect towards it. Therefore, we update the weights of each VNF due to Eq. 5 at each decision interval. This equation normalizes the weights of the existing VNFs between 0 to 100 making the relative weights to be readable for the load balancer.

$$weight_{VNF_i} = \left(\frac{100 * score^{-1}_{VNF_i}}{\sum_{VNF_j \in S_t} (score^{-1}_{VNF_j})}\right) \quad (5)$$

Algorithm 1 proposed joint load balancing and auto-scaling algorithm executed every t seconds.

1: Set $t =$ to next Scaling and LB update window.
2: **for** $VNF_i \in \mathcal{S}$ **do**
3: Calculate $Score_{VNF_i}(\mathcal{U}_{VNF_i}, \mathcal{P}_{VNF_i}, \mathcal{A}_{VNF_i})$
4: Update $State_{VNF_i}(Score_{VNF_i})$
5: Update $Weight_{VNF_i}(Score_{VNF_i})$
6: **end for**
7: $instances \leftarrow$ Number of current service instances in $\mathcal{S}$
8: $T_{score} = \sum_{VNF_i \in \mathcal{S}} \frac{Score_{VNF_i}}{instances}$
9: **if** $(<\forall State_{VNF_i}> = OL) \vee (T_{score} \geq Threshold_{average})$ **then**
10: trigger ScaleOut()
11: Update Loadbalancing()
12: **else if** $(<\forall State_{VNF_i}> = UL)$ **then**
13: trigger ScaleIn()
14: Update Loadbalancing()
15: **else**
16: **for** each $VNF_i \in \mathcal{S}$ **do**
17: update Loadbalancer($Weight_{VNF_i}$)
18: **end for**
19: **end if**

Let T_{score} denote the average score of all the existing VNFs. The scale out decision is triggered whenever T_{score} exceeds a predefined $Threshold_{average}$ or the majority of VNFs are in an *overloaded* state (lines 9-11). The latter ensures that a new instance or some *underloaded* existing VNFs do not decrease the $Threshold_{average}$ in a way that scaling policy misses a scale out decision even if the majority of VNFs have already reached their maximum capacity usage. The newly instantiated VNF is added to the load balancer's pool and the associated weight to this VNF is set to the minimum weight among all the existing VNFs in the last window. On the other hand, the scale-in decision is made only if the majority of the VNFs are in an *underloaded* state (lines 12-14). Subsequently, one VNF with the lowest score is removed from the load balancer's pool. It should be noted that the selected service instance during the scale-in procedure should not be deleted instantly. To avoid service disruption of the already assigned traffic to the VNF with the least score, we postpone the elimination of that VNF to the next window. During each iteration, if no scale-out/in is triggered, our proposed algorithm updates the load balancing weights of the service instances to reduce the traffic volume sent to the more loaded VNFs (lines 15-18).

5. Evaluation

In this section, we validate the efficacy of our proposed multi-criteria reactive approach for joint VNF load balancing and auto-scaling by comparing it to a real environment method in the state-of-the-art. We deploy both the proposed and baseline methods in the orchestration layer of NFV. Subsequently, we study the system's behaviour under different load patterns for both of these methods.

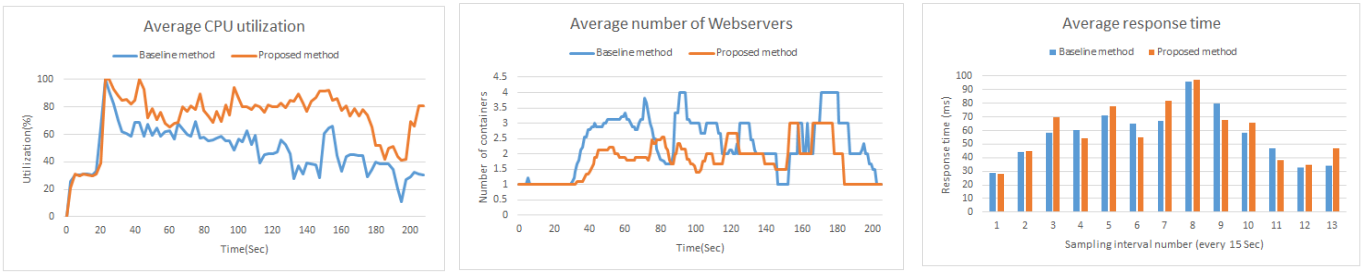
5.1. Experimental setup

We develop our proposed method in the orchestration layer of our previously introduced SDN/NFV testbed, in accordance with the proposed architecture in Fig. 2. The required interactions and communication are provided through XNFV agent and the orchestration components. The computing infrastructure registered to XNFV is a server equipped with 8 cores Intel Xeon E5-2620 v4 2.1 GHz CPUs and 32 GB of RAM. All the constituent components are deployed as Docker-based containers [37]. We utilize the monitoring system of the framework to obtain the required information for decision makings every 2.5 seconds. The elastic application used to test our proposed solution is a web server. The ingress traffic passes through an Openflow virtual switch and is redirected to the load balancer VNF by the SDN controller. All the elastic service instances are deployed in the above-mentioned computing infrastructure and are connected to a single Openflow virtual switch. To avoid extra overhead and unnecessary load balancer reconfigurations, we update the service instance weights every 15 seconds. Defining a smaller interval size for the weight updates would definitely improve the load balancing functionality, but it imposes extra overhead that may cause availability issues.

VNFs: We choose Apache Httpd [38] as our web server. All the service instances contain 4 sizes of web pages (1KB, 50KB, 100KB, 1MB). The load balancer VNF, which in our case is HAProxy [39], distributes the incoming web page requests among the existing service instances. We assume that the network of the NFVI-PoP is SDN-enabled. Therefore, the SDN controller VNF, which is OpenDayLight [40], routes the traffic through Openflow virtual switches in the server using the static flows that we set regarding our network topology. We generate the traffic requests by Httperf VNF [41] using the developed scripts for different scenarios explained in the next subsections.

Parameters and metrics*:* There are some parameters and metrics that have to be defined to fully implement the scenarios in an automated manner. The parameters include the coefficients used to calculate score function and each of the variables U, P, and A. Choosing the optimum value of the aforementioned weights are complicated, and it is totally dependent on the application and the chosen metrics within each utility function. We set α, β, γ, δ, and φ to 3, 1, 3, 1, and 1, respectively. Even though the efficiency of the proposed method is reliant on the parameters value, but obtaining appropriate results with arbitrary and reasonable values shows the superiority of our proposed method. Therefore, we delegate this task to service providers to optimize these values according to their service. In addition to the CPU and memory utilization, we set the number of busy threads of the web server as the application metric which is provided by Httpd. The response time of each web server instance is used as the performance metric. We set the upper threshold to 70% while the bottom threshold is 30%.

Traffic: To evaluate the proposed solution under different traffic patterns, we design two traffic scenarios using Httperf. The first scenario sends requests with an exponential distribution with incremental λ every 15 seconds, and after



(a) Average CPU utilization of running service instances.

(b) Average number of service instances.

(c) Service response time per sampling interval.

Figure 8: Comparison of the proposed and baseline method under incremental/decremental traffic.

120 seconds it decrements the rate with the same distribution. The second scenario aims to investigate the performance of the proposed algorithm in the case of receiving traffic with real case attributes. Therefore, we inject exponential distribution random number of calls per web request. The traffic pattern has an ON-OFF behavior with lognormal distribution for both durations of the ON/OFF and the packet inter-arrival times within the ON/OFF periods, as [42] and [43] suggest for generating realistic data center and web server traffic, respectively. In particular, the mean and sigma value of the underlying normal distribution for lognormal are set (2.5,0.5) and (1.8,0.4) for the duration of ON and OFF periods.

5.1. Results and comparison

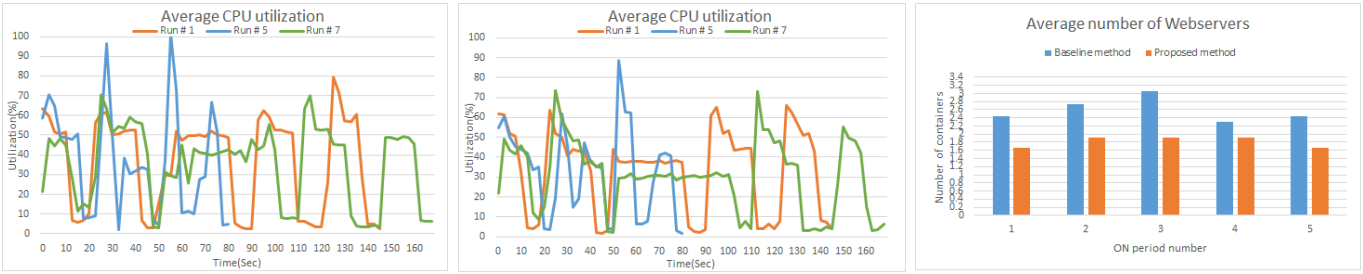
We compare the proposed multi-criteria reactive approach for joint VNF load balancing and service auto-scaling with the practical reactive approach adopted by most of the orchestration frameworks and industrial platforms that support container scaling (e.g., Kubernetes, Docker-swarm, Amazon ECS). We deploy a threshold-based auto-scaling using the average CPU utilization alongside a static round-robin load balancing algorithm in the orchestration layer of XNFV as the baseline solution. We run the two traffic scenarios for the target elastic service to monitor the system behavior while our proposed method and the baseline solution are deployed in turn. To provide a fair comparison, the characteristics of the generated traffic for each scenario of the proposed method are stored at the run-time by the developed scripts. In this way, the same traffic is regenerated using Httperf to observe the system's behavior having the baseline method deployed. To obtain valid run-time statistics, we run each traffic scenario 10 times with different starting traffic rates, calls per request, and distributions parameters. The following run-time system attributes are observed to cover different aspects of the evaluation.

Resource Utilization: The CPU and memory resource utilizations are monitored throughout each run. As the target application is more CPU-intensive, we only report the mean value of the average CPU utilization of the running service instances every 2.5 seconds over 10 run replications for the first traffic pattern. However, for the second traffic pattern, we show only the average CPU utilization of three running scenarios individually since the duration of ON-OFFs and the execution time for each scenario are different from one another. As depicted in Fig. 8 (a), Fig. 9 (a), and (b), the average resource usage for the baseline method is lower than the proposed method which implies that the resources are managed less efficiently. The proposed method provides a better resource utilization balance among running service instances at each time for both traffic patterns due to better standard deviation of the CPU utilization. This is achieved through the dynamic weight update in the load balancer based on the score value of each service instance that is also used in scaling decision making. It should be noted that the scaling engine in both methods creates heterogeneous new instances. Therefore, the round-robin load balancing algorithm in the baseline method would cause even more extreme imbalance of the resource utilization if the instances have different processing capacity compared to the auto-scaling aware load balancing of our proposed method.

In summary, even though the baseline method has less average resource utilization, the proposed method provides a better load balance between service instances. Moreover, the proposed method considers application run-time status and requests response time in both scaling and load balancing decisions which results in more efficient use of the allocated resources.

Cost: We assume that the cost of the joint VNF load balancing and service auto-scaling is significantly relative to the number of scale out/in and the number of running instances at each time. From a resource management perspective, it is important that the system guarantee the desirable QoS with less allocated resources. Fig. 8 (b) and Fig. 9 (c) perfectly demonstrate that the proposed method instantiates a much smaller number of service instances for both traffic scenarios during the run-time. In the case of ON-OFF traffic, we only report the number of running instances during the ON periods as within the OFF periods much less load is imposed on the system and only a limited number of instances are required. However, the proposed method would have fewer running instances at the beginning of each OFF period before scaling in occurrences. The proposed method avoids unnecessary scale outs by also taking into account the application functional status and incoming requests response time, unlike the baseline method which only takes CPU resource status into consideration. Hence, the baseline method would scale out upon sudden momentary increase of CPU utilization due to incoming traffic rate change. This shows that our proposed method is more resilient to the burst traffic as the number of errors in the next subsection verifies this claim.

Service response time and error: The response time of the elastic service is a proper metric to evaluate our orchestration



(a) Average CPU utilization of three runs for proposed method.

(b) Average CPU utilization of three runs for baseline method.

(c) Average number of service instances during ON periods.

Figure 9: Comparison of the proposed method and baseline method under ON-OFF traffic.

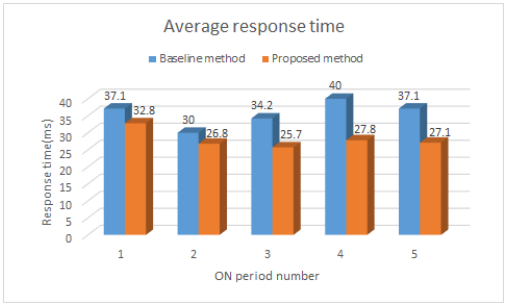
layer functionalities. Fig. 8 (c) shows the average service response time in each interval for incremental/decremental traffic. During the intervals that the running service instances are not saturated, the average response time of the service is almost the same. In some intervals, the response time of the baseline method is slightly better than the proposed method as the number of associated service instances is considerably more and the traffic pattern is incremental and even the unnecessary scale outs would be beneficial in the near future when the incoming traffic rate increases. However, during these intervals, the average response time has not exceeded the tolerable response time threshold set in the proposed method (75ms). Consequently, the proposed method does not trigger the scale out process. Nevertheless, the average response time of the proposed method is less than the baseline method in some intervals for several reasons. First, the load is more balanced between service instances when using the proposed method. Second, sometimes the response time exceeds the tolerable value even if the CPU utilization does not reach the upper threshold. In this case, the proposed method triggers scale out while the baseline method misses the required scale out decision. Finally, the load balancing algorithm and scaling policy of our proposed method work aware of one another through the common score and the dynamic weight update.

On the other hand, Fig. 10 (a) shows the average response time of the service during the ON periods of the ON-OFF traffic. As it can be seen, the proposed method outperforms the baseline method significantly since the scaling decisions made during one ON period do not affect the system behavior in the next ON period. Moreover, the average number of errors caused by the proposed method is much less than the baseline method (Fig. 10 (b)). These errors mostly happen because of request time outs which occur when the response time exceeds 2 seconds (set in traffic scenario). Consequently, considering these over limit response times would even increase the superiority of the proposed method in terms of system performance in Fig. 10 (a).

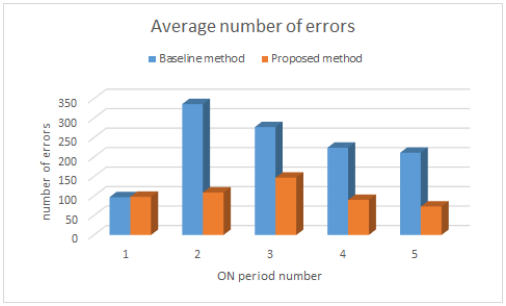
6. Conclusion and future work

In this paper, we demonstrated that the load balancing among VNF instances of an elastic service and the auto-scaling policy must be designed aware of one another. We show that providing such awareness in the orchestration functionalities of NFV improves the quality of decision making that results in better QoS and resource management. This is critical especially when leveraging NFV in different locations of the 5G networks such as RAN, mobile core network, edge and cloud. We conducted several real environment experiments to identify the key factors for the coordination of the load balancing and auto-scaling using the multimedia case study. Consequently, we proposed a novel multi-criteria reactive approach for joint dynamic VNF load balancing and service auto-scaling using a mutual score function to match the scaling decisions with dynamic load balancing weights. We evaluated the proposed method under different real environment traffic patterns with valid traffic distribution attributes. The results show that the proposed solution outperforms existing practical solutions adopted by commercial platforms in terms of resource management, service performance, cost and burst traffic resiliency. It is worth noting that we developed the proposed operational units of the automated orchestration on top of the XeniumNFV, open source testbed. Hence, the practical challenges of deploying an efficient framework for service auto-scaling and load balancing are precisely considered.

In the future, we aim to devise a novel joint load balancing and auto-scaling solution with a hybrid reactive and proactive approach using machine learning. We will solve the joint optimization problem by taking into account cost and performance constraints.



(a) Average response time during ON periods.



(b) Average number of errors during ON periods.

Figure 10: Service response time and error comparison of the proposed and baseline method under ON-OFF traffic.

References

[1] A. Kusedghi, A. Ghorab, and A. Akbari, "XeniumNFV: A unified, dynamic, distributed and event-driven SDN/NFV testbed," Proc. Of Int'l. Conf. on Cloud Computing Technology and Science (CloudCom), Nicosia, Cyprus, Dec., pp. 320–326, 2018.

[2] S. H. K. S. L. Chen, Y. Chen, "CLB: A novel load balancing architecture and algorithm for cloud services," *Computers and Electrical Engineering*, vol. 58, no. 1, pp. 154–160, Feb. 2017.

[3] D. J. A. Singh and M. Malhotra, "Autonomous agent based load balancing algorithm in cloud computing," Proc. Computer Science, vol. 45, Jan., pp. 832–841, 2015.

[4] C. C. Li and K. Wang, "An SLA-aware load balancing scheme for cloud datacenters," Proc. of Int'l. Conf. on Information Networking (ICOIN), Phuket, Thailand, Feb., pp. 58–63, 2014.

[5] S. M. B. A. Akbar and A. Sattar, "A Comparative Study on Load Balancing Algorithms for SIP Servers," New Delhi: Springer, Feb., pp. 79–88, 2016.

[6] V. Nguyen, K. Grinnemo, J. Taheri, and A. Brunstrom, "On load balancing for a virtual and distributed MME in the 5G core," Proc. of Int'l. Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), Bologna, Italy, Sept. 2018.

[7] M. Thai, Y. Lin, and Y. Lai, "A joint network and server load balancing algorithm for chaining virtualized network functions," Proc. of Int'l. Conf. on Communications (ICC), Kuala Lumpur, Malaysia, May 2016.

[8] X. T. W. Chen, Z. Shang and H. Li, "Dynamic server cluster load balancing in virtualization environment with Openflow," *International Journal of Distributed Sensor Networks*, vol. 11, no. 7, pp. 531–538, July 2015.

[9] W. Zhang, T. Wood, and J. Hwang, "Netkv: Scalable, self-managing, load balancing as a network function," Proc. of Int'l. Conf. on Autonomic Computing (ICAC), Wurzburg, Germany, July 2016.

[10] S. J. Y. Go, R. Guinto, C. A. M. Festin, I. Austria, R. Ocampo, and W. M. Tan, "An SDN/NFV-enabled architecture for detecting personally identifiable information leaks on network traffic," Proc. of Int'l. Conf. on Ubiquitous and Future Networks (ICUFN), Zagreb, Croatia, July 2019.

[11] M. Abdullah, W. Igbal, and F. Bukhari, "Containers vs virtual machines for auto-scaling multi-tier applications under dynamically increasing workloads," Proc. of Int'l. Conf. on Intelligent Technologies and Applications (INTAP), Bahawalpur, Pakistan, Oct. 2019.

[12] A. Kwan, J. Wong, H. Jacobsen, and V. Muthusamy, "Hyscale: Hybrid and network scaling of dockerized microservices in cloud data centres," Proc. of Int'l. Conf. on Distributed Computing Systems (ICDCS), Dallas, TX, July 2019.

[13] F. Rossi, M. Nardelli, and V. Cardellini, "Horizontal and vertical scaling of container-based applications using reinforcement learning," Proc. of Int'l. Conf. on Cloud Computing (CLOUD), Milan, Italy, July 2019.

[14] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-scaling web applications in clouds: A taxonomy and survey," *ACM Comput. Surv*., vol. 51, no. 4, Sept. 2018.

[15] R. Mijumbi, S. Hasija, S. Davy, A. Davy, B. Jennings, and R. Boutaba, "Topology-aware prediction of virtual network function resource requirements," *IEEE Trans. On Network and Service Management*, vol. 14, no. 1, pp. 106–120, Mar. 2017.

[16] V. Sciancalepore, F. Z. Yousaf, and X. Costa-Perez, "z-torch: An automated NFV orchestration and monitoring solution," *IEEE Trans. On Network and Service Management*, vol. 15, no. 4, pp. 1292–1306, Dec. 2018.

[17] J. Duan, C. Wu, F. Le, A. X. Liu, and Y. Peng, "Dynamic scaling of virtualized, distributed service chains: A case study of IMS," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2501– 2511, Nov. 2017.

[18] A. M. Medhat, G. A. Carella, M. Pauls, and T. Magedanz, "Extensible framework for elastic orchestration of service function chains in 5g networks," Proc. of Int'l. Conf. on Network Function Virtualization and Software Defined Networks (NFV-SDN), Berlin, Germany, pp. 327–333, Nov. 2017.

[19] F. Lombardi, A. Muti, L. Aniello, R. Baldoni, S. Bonomi, and L. Querzoni, "Pascal: An architecture for proactive auto-scaling of distributed services," *Future Generation Computer Systems*, vol. 98, no. 2, pp. 342 – 361, Sept. 2019.

[20] W. Iqbal, A. Erradi, M. Abdullah, and A. Mahmood, "Predictive auto-scaling of multi-tier applications using performance varying cloud resources," *IEEE Transactions on Cloud Computing* (Early Access, pp. 1–1, Sept. 2019.

[21] I. Alawe, Y. Hadjadj-Aoul, A. Ksentinit, P. Bertin, C. Viho, and D. Darche, "An efficient and lightweight load forecasting for proactive scaling in 5G mobile networks," Proc. of Int'l. Conf. on Standards for Communications and Networking (CSCN), Paris, France, Oct. 2018.

[22] D. Cotroneo, R. Natella, and S. Rosiello, "NFV-throttle: An overload control framework for network function virtualization," *IEEE Trans. On Network and Service Management*, vol. 14, no. 4, pp. 949–963, Dec. 2017.

[23] R. N. D. Cotroneo and S. Rosiello, "Overload control for virtual network functions under CPU contention," *Future Generation Computer Systems*, vol. 99, no. 1, pp. 164–176, Oct. 2019.

[24] A. Bauer, N. Herbst, S. Spinner, A. Ali-Eldin, and S. Kounev, "Chameleon: A hybrid, proactive auto-scaling mechanism on a levelplaying field," *IEEE Trans. on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 800–813, Apr. 2019.

[25] A. Bauer, V. Lesch, L. Versluis, A. Ilyushkin, N. Herbst, and S. Kounev, "Chamulteon: Coordinated auto-scaling of micro-services," Proc. Of Int'l. Conf. on Distributed Computing Systems (ICDCS), Dallas, TX, Oct. 2019.

[26] B. Benifa and D. Dharma, "HAS: Hybrid auto-scaler for resource scaling in cloud environment," *Journal of Parallel and Distributed Computing*, vol. 120, no. 1, pp. 1–15, Oct. 2018.

[27] L. Velasco, R. Casellas, and S. Llana, "A control and management architecture supporting autonomic NFV services," *Photonic Network Communications*, vol. 37, no. 1, pp. 24–37, Feb. 2019.

[28] R. Moreira, F. de Oliveira Silva, P. Frosi Rosa, and R. Aguiar, "A flexible network and compute-aware orchestrator to enhance QoS in NFV-based multimedia services," Proc. of Int'l. Conf. on Advanced Information Networking and Applications (AINA), Krakow, Poland, pp. 512–519, May 2018.

[29] R. Poddar, A. Vishnoi, and V. Mann, "Haven: Holistic load balancing and auto scaling in the cloud," Proc. of Int'l.

Conf. on Communication Systems and Networks (COMSNETS), Bangalore, India, Jan. 2015.
[30] W. Abderrahim and Z. Choukair, "Dependability integration in cloud-hosted telecommunication services," *IEEE Trans. on Dependable and Secure Computing*, vol. 16, no. 6, pp. 957–968, Nov. 2019.
[31] A. J. Gonzalez, G. Nencioni, A. Kamisinski, B. E. Helvik, and P. E. ´ Heegaard, "Dependability of the NFV orchestrator: State of the art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 20, no. 4, pp. 3307 3329, Fourthquarter 2018.
[32] A. Kusedghi, Z. Bagherabadi, and A. Akbari, "An IMS-aware VM placement in cloud environment," Proc. of Int'l. Conf. on Cloud Computing Technology and Science (CloudCom), Nicosia, Cyprus, pp. 327–334, Dec. 2018.
[33] "Sipp," https://github.com/SIPp/sipp, Sept. 2020, accessed on 2020-09- 01.
[34] Sangoma Technologies, "Asterisk," https://www.asterisk.org/, Sept. 2020, accessed on 2020-09 01.
[35] OpenSIPS project, "Opensips," https://opensips.org/, July 2020, accessed on 2020-09-01.
[36] Kamailio SIP project, "Kamailio sip load balancer," https://www.kamailio.org/w/, Sept. 2020, accessed on 2020-09-01.
[37] Docker, "Docker hub," https://www.docker.com/, Sept. 2020, accessed on 2020-09-01.
[38] Apache Software Foundation, "Apache httpd," https://httpd.apache.org/, Apr. 2020, accessed on 2020-09-01.
[39] HAProxy community edition, "Haproxy," http://www.haproxy.org/, Dec. 2019, accessed on 2020-09-01.
[40] The Linux Foundation, "Opendaylight," https://www.opendaylight.org/, Dec. 2018, accessed on 2020 09-01.
[41] D. Mosberger and T. Jin, "Httperf—a tool for measuring web server performance," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 3, p. 31–37, Dec. 1998.
[42] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, p. 92–99, Jan. 2010.
[43] Z. Liu, N. Niclausse, and C. Jalpa-Villanueva, "Traffic model and performance evaluation of web servers," *Performance Evaluation*, vol. 46, no. 2, pp. 77 – 100, Oct. 2001.

Amir Kusedghi received his BSc and MSc degrees in Computer engineering from Shahid Beheshti University and Sharif University of Technology, in 2010 and 2013 respectively. He is currently Ph.D. candidate and his research interests include Computer networks and security, SDN, NFV, and 5G networks.

Email: kusedghi@comp.iust.ac.ir

Ahmad Akbari received the B.Sc. degree in electronics engineering and the M.Sc. degree in communications engineering from IUT, Isfahan, Iran, in 1987 and 1989, respectively, and the Ph.D. degree in electrical engineering from Rennes 1 University Rennes, Rennes, France, in 1995.
He is an Associate Professor with the School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran. He has been the Dean of the IUST School of Computer Engineering from 2013 to 2020. His research interests include computer networks security, data communications, and signal processing applications.
.

Email: akbari@iust.ac.ir

Paper Handling Data:

Submitted: 11-01-2020
Received in revised form: 05-10-2021
Accepted: 09-24-2021
Corresponding author: Ahmad Akbari
Affiliation of the corresponding author: School of Computer Engineering, Iran University of Science and Technology and Center of Excellence in Future Networks, Iran University of Science and Technology, Tehran, Iran.