Predictive Auto-scaling with OpenStack Monasca

Giacomo Lanciano
Scuola Normale Superiore
Pisa, Italy
giacomo.lanciano@sns.it

Filippo Galli
Scuola Normale Superiore
Pisa, Italy
filippo.galli@sns.it

Tommaso Cucinotta
Scuola Superiore Sant'Anna
Pisa, Italy
tommaso.cucinotta@santannapisa.it

Davide Bacciu
University of Pisa
Pisa, Italy
davide.bacciu@di.unipi.it

Andrea Passarella
National Research Council
Pisa, Italy
andrea.passarella@iit.cnr.it

ABSTRACT

Cloud auto-scaling mechanisms are typically based on *reactive* automation rules that scale a cluster whenever some metric, e.g., the average CPU usage among instances, exceeds a predefined threshold. Tuning these rules becomes particularly cumbersome when scaling-up a cluster involves non-negligible times to bootstrap new instances, as it happens frequently in production cloud services.

To deal with this problem, we propose an architecture for auto-scaling cloud services based on the status in which the system is expected to evolve in the near future. Our approach leverages on time-series forecasting techniques, like those based on machine learning and artificial neural networks, to predict the future dynamics of key metrics, e.g., resource consumption metrics, and apply a threshold-based scaling policy on them. The result is a *predictive* automation policy that is able, for instance, to automatically anticipate peaks in the load of a cloud application and trigger ahead of time appropriate scaling actions to accommodate the expected increase in traffic.

We prototyped our approach as an open-source OpenStack component, which relies on, and extends, the monitoring capabilities offered by Monasca, resulting in the addition of predictive metrics that can be leveraged by orchestration components like Heat or Senlin. We show experimental results using a recurrent neural network and a multi-layer perceptron as predictor, which are compared with a simple linear regression and a traditional non-predictive auto-scaling policy. However, the proposed framework allows for the easy customization of the prediction policy as needed.

KEYWORDS

Elasticity auto-scaling, Time-series forecasting, Predictive operations, OpenStack

ACM Reference Format:
Giacomo Lanciano, Filippo Galli, Tommaso Cucinotta, Davide Bacciu, and Andrea Passarella. 2021. Predictive Auto-scaling with OpenStack Monasca . In *14th IEEE/ACM International Conference on Utility and on Cloud Computing (UCC 2021), December 6-9, 2021.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Information and communications technologies have been undergoing a steep change over the last decade, where the massive availability of low-cost high-bandwidth connectivity has brought to a tremendous push towards distributed computing paradigms. This has led to the raise of cloud computing technologies [7], where the developments in a number of virtualization technologies, spanning across the computing, networking and storage domains, has facilitated the decoupling between management of the physical infrastructure from the cloud services provisioned on top of them. In the last decade, cloud computing has evolved from infrastructure-as-a-service (IaaS) provisioning scenarios, where physical servers were simply migrated within virtual machines (VMs), to the nowadays platform-as-a-service (PaaS) era in which *native* cloud applications and services are developed relying heavily on a plethora of networking, storage, security, load-balancing, monitoring and orchestration services (XaaS - everything-as-a-service) made available within cloud infrastructures and their automation capabilities [6].

A key success factor of cloud computing relies on the ability of cloud providers to manage the infrastructure 24/7, promptly addressing and resolving any issue that may occur at run-time, including both hardware faults and possible software malfunctioning. This is made possible by the use of appropriate data-center designs (i.e., fault-independent zones, redundant powering and cooling infrastructures and multi-path networking topologies) coupled with the use of feature-rich resource managers and orchestrators.

A *cloud orchestrator* is the software at the center of that concept of "rapid provisioning" with "minimal management effort or service provider interaction" from the quite popular cloud computing definition from NIST [21]. It includes key automation features that give a cloud infrastructure self-healing and self-management capabilities, thanks to the deployment of fine-grained monitoring infrastructures that allow for triggering a number of automation rules, to deal with a plethora of different issues: from handling automatically hardware faults (individual computing, networking or storage elements failing), to the automatic resizing of elastic and horizontally scalable services, in order to cope with dynamic traffic conditions and mitigate possible overloading conditions. A quite popular cloud orchestrator is for example the open-source OpenStack software.

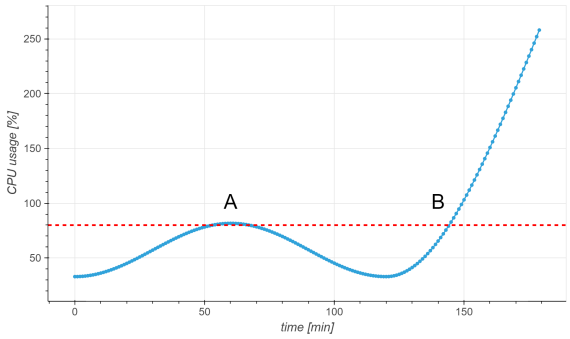


Figure 1: Example of load profile.

The "elasticity" of cloud services, namely their capability to expand their deployment over more and more instances (VMs, containers or physical services if allowed) as the workload increases, allows for keeping a stable performance (response times observed by individual clients) despite fluctuations of the overall aggregated traffic towards the services. This is realized through the use of control loops, where an elasticity controller may decide to instruct scale-out or scale-in operations, based on monitoring observations made over time for individual services. These decisions are typically made on the basis of monitored infrastructure-level resource consumption metrics (e.g., CPU, networking, storage load), as well as application-level metrics (e.g., response times, connection errors, timeouts).

Traditionally, elasticity loops are performed applying a number of *threshold-based* rules: when a given metric (or a combination of metrics) exceeds some warning or critical threshold, a scale-out operation is triggered (e.g., resource utilization beyond certain limits or service latency beyond acceptable values). Also, in order to avoid continuous scaling operations due to the inherent fluctuations of the monitored metric(s), it is typical to have a high-water/low-water double-threshold set-up, where a scale-up is performed when the monitored metric grows beyond a high threshold, whilst a scale-down is performed when it returns below a much lower threshold. Moreover, in order to avoid scale-up operations on transient metric peaks, it is commonplace to require a violation of the threshold for a few consecutive observations (typically, 3 samples or 5 samples), before triggering the action.

Designing such seemingly simple control loops is accompanied by a set of challenges, such as: determining the most effective key performance indicators (KPIs) and the frequency at which they should be monitored; tuning the scaling policy so that it can quickly adapt to substantial changes in the application workloads, while being robust to fluctuations; estimating how many and which type of resources are required in order to handle the new conditions; deciding which scaling strategy (e.g., horizontal or vertical) is best suited.

Despite the mentioned precautions, control loops remain fundamentally "dumb", in the sense that they do not account for the rich dynamic of the observed metrics. As a simple example, consider the conceptual metric evolution depicted in Figure 1. Here, we can see two scenarios where a metric starts growing at quite different rates. A threshold-based scale-out would treat these two conditions in pretty much the same way. However, a human operator observing these curves, understands quite easily that scenario B needs a quicker and more urgent action than scenario A. Although, this depends also exactly on the time needed to bring up a new instance.

In the depicted scenario, a fundamental role may be played by metric prediction and forecasting techniques, which, if properly integrated within the automation rules of a cloud orchestrator, may constitute a powerful tool to embed more intelligence within elasticity controllers.

1.1 Contributions

This paper proposes an architecture for integrating predictive analytics within an OpenStack cloud orchestration engine. The contributions of this paper include: 1) a general architecture for performing *predictive operations* on a cloud infrastructure, based on smart metric forecasts that can be obtained via machine learning (ML) or artificial intelligence (AI); 2) an open-source implementation of the metric forecast component within OpenStack, leveraging on the Monasca monitoring architecture, that automatically computes metric forecasts making them available as additional metrics that can be leveraged within the system at any level; 3) a few simple metric predictors implemented in the framework as customizable predictors based on linear regression (LR), multi-layer perceptrons (MLPs) and recurrent neural networks (RNNs); the proposed architecture is expandable, so it eases the implementation of additional predictive models in the code; 4) results from real experimentation on a simple use-case based on synthetic workload where we exploit the proposed architecture to create *predictive elasticity rules* using Monasca and Senlin, exploiting the native capabilities of OpenStack so to realize intelligent automation rules.

2 RELATED WORK

ML and AI techniques have been investigated for a number of tasks related to resource management in cloud computing infrastructures, in the context of elasticity for both general public cloud services and private cloud infrastructures. In the following, we provide an overview of key research papers dealing with intelligent elasticity management based on metric forecasting for general public cloud services. Then we present related research in predictive autoscaling for private cloud infrastructures with a focus on network function virtualization (NFV) [24] and deployment of distributed elastic service chains. Finally, we include a short review of elasticity control solutions based on learning methods, such as reinforcement learning (RL).

2.1 Predictive elasticity in cloud computing

In [5], a multi-layer neural network has been used to predict the resource usage of tasks performing continuous integration

of several repositories from the Travis openly available data. Using a per-repository trained model, an accuracy at least 20% and up to 89% better than a baseline linear regression has been achieved.

In [37], model-predictive techniques have been used to track and predict the workload variability so to optimize resource allocation in elastic cloud applications. The proposed technique leverages an ARMA model for workload prediction, and it tries to balance the advantages arising from dynamic elasticity, to the cost due to applying the scaling decisions and reconfiguring the cluster at each control period.

More recently, in the RScale framework [15], Gaussian Process Regression, a probabilistic machine-learning method, has been used to predict end-to-end tail-latency of distributed micro-services workflows with generic DAG-alike topologies. This was evaluated on a NSF Chamaleon test-bed, achieving similar accuracy but a smaller predicted uncertainty with respect to using neural networks, but with greatly enhanced execution-time overheads, and greater capability to adapt on-line to dynamically changing workload/interference conditions.

In [13], the problem of non-instantaneous instance provisioning when using elastic scaling in cloud environments is tackled, by proposing a predictive scaling strategy based on a resource prediction model using neural networks and linear regression. The method has been applied on an e-commerce application scenario emulated through the well-established TPC-W [40] workload generator and benchmarking application, deployed within AWS EC2. Predictions made via neural networks enhance the accuracy by reducing the mean average percentage error (MAPE) by roughly 50% compared to linear regression.

In [4], Bayesian Networks are used in a predictive framework to support automatic scaling decisions in cloud services. However, the method is evaluated on synthetic applications with exponential start times, duration and workload inter-arrival patterns.

Other approaches exist that prefer to focus on real-time dynamic resource allocation based on instantaneous monitoring rather than resource estimations/predictions made ahead of time. For example, in [25], a vertical elasticity management of containers has been proposed, to adapt dynamically the memory allocated to containers in a Kubernetes cluster, so to better handle the coexistence of containers with heterogeneous quality of service (QoS) requirements (guaranteed, burstable and best-effort). However, approaches of this type fall within the research literature on classical reactive elasticity loops, which we omit for the sake of brevity.

2.2 Predictive elasticity for NFV services

Predictive techniques in private cloud computing scenarios have also been investigated in the context of NFV and Software Defined Networks (SDN) for adapting available virtualized resources to varying loads [19, 23, 26]. This way, operators can benefit from proactive automation mechanisms to ensure appropriate QoS for their cloud-native *service-chains*. Although, several challenges must be tackled in order to obtain an effective solution to such problem. For instance: (i) correctly assessing which components need to be scaled to remove bottlenecks; (ii) optimizing the consolidation of virtual resources on the physical infrastructure; (iii) designing effective predictive models that prevent virtual resources from being under- or over-provisioned. [11].

The authors of [35] describe the usage of *ensembling* techniques (i.e., combining the outputs from several models) for auto-scaling purposes. In [43], instead, the authors propose a different approach based on Long Short-Term Memory (LSTM) networks, a particular flavor of RNNs, that is used for virtual network function (VNF) demand forecasting. Nowadays, RNNs have been proven to be a powerful tool for time-series analysis, being them forecasting [10, 18, 36] or classification [14, 20] tasks. In particular, the *sequence-to-sequence* architectural pattern, widely adopted for machine translation and Natural Language Processing (NLP) tasks, yields surprisingly good results [38]. Such architectures typically consist of two distinct modules, the *encoder* and the *decoder*. In the context of NFV, sequence-to-sequence models can be used to capture the complex relationships between VNF metric sequences and infrastructure metric sequences [10]. Notice that, since the information exchange between the encoder and the decoder is restricted to the hidden state values, one can even design a model that uses different metrics for inputs and outputs.

Forecasting accuracy can also be boosted by additional information about the topology of the deployed VNFs, e.g., graph-like diagrams depicting the interactions among the VMs belonging to the same VNFs [23, 26]. For instance, the authors of [22] propose a *topology-aware* time-series forecasting approach leveraging on graph neural networks (GNNs) [3].

2.3 Elasticity control with Reinforcement Learning

The most widely used approach to elasticity control is *static thresholding* [8]. Even though it is a straightforward heuristic, it can yield surprisingly good results when dealing with simple systems. However, in general, threshold policies require careful tuning for each service whose elasticity must be controlled, making a generic approach impossible to be adopted in practice (as it would eventually lead to over- or under-provisioning). To overcome such inconveniences, several *dynamic thresholding* mechanisms have been proposed to adapt thresholds to the current conditions of the system. Such methods can be implemented with AI-based techniques as well, like RL [1, 2]. However, RL algorithms usually come with demanding computing requirements that often limit their applicability in real infrastructures. For instance, the authors of [39] propose a Q-learning-based algorithm to be used in an actual telco system. However, the developed agent is allowed to take several unexpected decisions before converging to the optimal policy. This is clearly not desired when deploying the system in a production environment. On the

other hand, the authors of [42] propose a rather successful RL-based approach to VNF service-chains deployment that jointly minimizes operation costs and maximizes requests throughput, also taking into account different QoS requirements.

3 BACKGROUND

This section contains background concepts useful for a better understanding of the approach proposed in Section 4. A number of key components of OpenStack architecture are detailed below, with reference to Figure 2. Additionally, we include some well-known definitions around RNNs, which have been used in our proposed architecture.

3.1 Nova, Glance and Cinder

Nova [32] is the OpenStack project providing the necessary tools to provision and manage compute instances. It supports the creation of VMs, bare metal servers and containers (limited). It leverages on the Glance [28] service for image management and provisioning, and the Cinder [27] service for management of block storage that can be used to mount remote volumes in VM instances. The Nova architecture consists in a number of server processes, that communicate each other via an RPC message passing mechanism, and a shared central database. The core of the architecture is the *compute* process, whose job is to manage the underlying hypervisor exploiting the capabilities of libvirt. The *compute* process communicates with the Nova database by proxying its queries via the *conductor* process. The *scheduler* process instead is responsible for interfacing with the compute instances placement service, whose behavior can be customized through a number of plugins (filters).

3.2 Neutron

Neutron [31] is the OpenStack component in charge of providing network-as-a-service connectivity among instances managed by Nova. It allows for managing and customizing per-customer dedicated virtual networks with their own numbering and DHCP configurations, and that can be enriched with security capabilities including management of firewall rules and virtual private networks (VPNs). Neutron used to include also load-balancing-as-a-service capabilities, which have recently been engineered into the separate Octavia service (see below).

3.3 Monasca

Monasca [30] is the OpenStack project providing an advanced monitoring-as-a-service solution that is multi-tenant, highly scalable, performant, and fault-tolerant. It consists in a microservice architecture, where each module is designed to play a well-defined role in the overall solution (e.g., a streaming alarm engine leveraging on Apache Storm, a notification engine, a persistence layer backed by an efficient time-series database). The core of the architecture is a Kafka message queue, that enables asynchronous communication among the components. Monasca also includes an agent module, that is distributed on the machines hosting the compute entities and is responsible for actually collecting the metrics and forwarding them to the message queue through the REST APIs.

Apart from being usable in an OpenStack deployment, Monasca can also be deployed in a Kubernetes environment as a standalone monitoring solution.

3.4 Senlin

Senlin [34] is the OpenStack project providing the necessary tools to create and operate easily clusters of homogeneous resources exposed by other OpenStack services. The interactions between Senlin and the other OpenStack resources is enabled by the *profile* plugins. Once a profile type is chosen (e.g., a Nova instance), a *cluster* of resources which the profile refers to can be created and associated with *policies*. Such objects define how the resources belonging to a cluster must be treated under specific conditions. For instance, one can define a *scaling* policy to automatically resize the cluster, as well as a *health* policy to replace the resources as needed, or a *load-balancing* policy to evenly distribute the workload.

Notice that, with respect to Heat's auto-scaling group abstraction, Senlin provides a more fine-grained control over the management policies that should be applied, as well as effective operation support tools. Indeed, for instance, Senlin is used at large companies like Blizzard Entertainment to provide their on-premise gaming servers with auto-scaling capabilities [41].

3.5 Octavia

Octavia [33], previously known as *Neutron LBaaS*, is the OpenStack project providing a scalable and highly-available load-balancing solution. Load-balancing is fundamental feature for the cloud, as it enables a number of other properties (e.g, elasticity, high-availability) that are considered of the utmost importance for a modern production cloud environment. Octavia delivers its services by managing a horizontally-scalable pool of Nova instances (VMs, bare metal servers or containers), known as *amphorae*, that leverage on the features provided by HAProxy. Such pool is orchestrated by the *controller*, that consists in a number of sub-components whose jobs include handling API requests and ensuring the health of the amphorae.

3.6 Recurrent Neural Networks

RNNs are commonly used for predicting uni-variate or multi-variate time-series evolution in the future. In this paper, we used only uni-variate predictions. Here, we recall basic concepts about how RNNs are trained and used, for the sake of completeness.

As widely known [12], when predicting a target metric evolution using an RNN model, future metric estimations are based on the current metric value and an H-dimensional vector, computed recurrently from the past I samples, representing its state. The evolution of the model is then governed

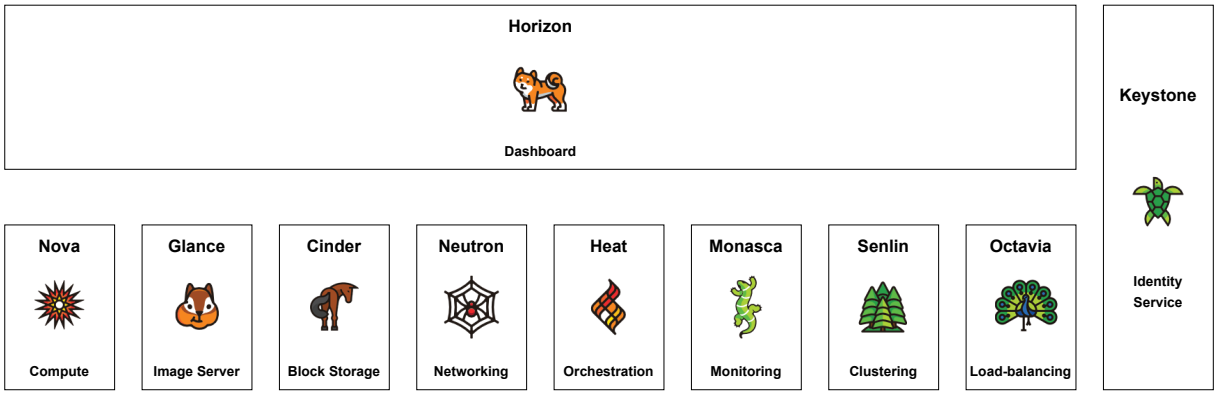


Figure 2: Overview of OpenStack key components.

by the following equations:

$$s_t = f_s(s_{t-1}, i_t) \tag{1}$$

$$o_t = f_o(s_t, i_t) \tag{2}$$

where $f_s(s,i) : \mathbb{R}^{H+I} \rightarrow \mathbb{R}^H$ acts on x, the concatenation of the hidden state vector $s \in \mathbb{R}^H$ and input vector $i \in \mathbb{R}^I$, and $f_s = \text{ReLU}(W_s x + b_s)$, with W_s and b_s being the weight and bias tensors respectively, and ReLU denoting the element-wise Rectified Linear Unit activation function; $f_o : \mathbb{R}^{H+I} \rightarrow \mathbb{R}^O$ is the output function defined as $\text{ReLU}(W_o x + b_o)$. Training of the learnable parameter set $\theta = \{\theta_j\} = \{W_s, W_o, b_s, b_o\}$ is achieved through the gradient descent algorithm with momentum, so that at the k-th optimization step the j-th parameter will be updated according to:

$$\mu_{j,k} = \beta \mu_{j,k-1} + \nabla J_{\theta_{j,k}}(D) \tag{3}$$

$$\theta_{j,k+1} = \theta_{j,k} - \lambda \mu_{j,k} \tag{4}$$

With $\nabla J_{\theta_{j,k}}(D)$ being the gradient of the cost function $J_{\theta_{j,k}}(D)$ computed with respect to parameter θ_j at instant k over dataset D of input-output pairs, λ is the learning rate, and β is a hyper-parameter determining how much momentum $\mu_{j,k}$ is applied during the gradient descent step, usually set to 0.9. Training continues until convergence, i.e. when the minimum value of the loss function has been reached. A working definition of *minimum value* involves computing the loss function over a held-out validation dataset D' every K optimization steps, and continuing training as long as the validation loss keeps decreasing. As the validation loss curves up again (as a consequence of over-fitting) the optimal model is taken as the model corresponding to the minimum of the validation loss.

4 PROPOSED APPROACH

Our approach consists in a *predictive* auto-scaling strategy, that estimates the future state of the monitored system and takes scaling decisions based on such estimates. Depending on the quality of the estimates, such a strategy is able, for instance, to automatically anticipate peaks in the load of a cloud application and trigger appropriate scaling actions to accommodate the increasing traffic. This feature allows for overcoming the main drawbacks of classical—*reactive*—auto-scaling approaches, that typically start acting on the system when the load is already too high to be properly handled, and the QoS is already degraded. Of course, operation teams could err on the side of caution and tune the system such that the auto-scaling kicks in well in advance with respect to the actual load peak. However, doing this entails paying additional costs for provisioning resources that could be potentially not needed. Resource over-provisioning is not a sustainable strategy in many scenarios. In the context of cloud applications, where certain components are typically designed to be horizontally scaled to accommodate a varying level of traffic, one should also take into account that spinning up new replicas can take a non-negligible amount of time. One of the key benefits of using a predictive approach is indeed the possibility to mitigate such a *cold-start* effect, by triggering the resource provisioning process at the right time

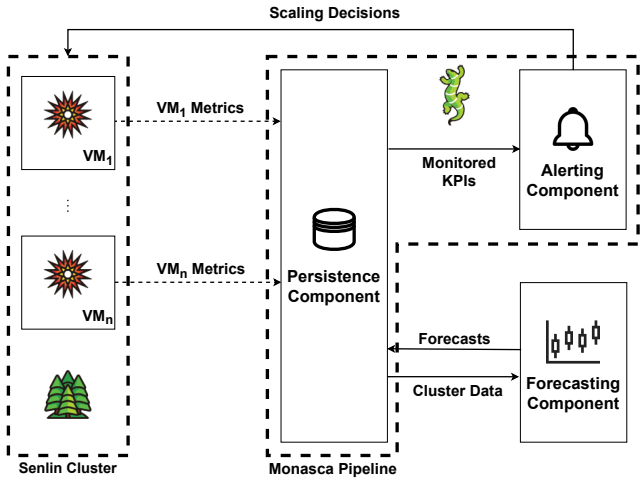


Figure 3: Architectural diagram of the proposed predictive auto-scaling approach.

for the replica to be ready when the additional traffic starts hitting the service.

To demonstrate the effectiveness of our approach, and its relevance in the context of modern cloud operations, we realized an implementation that is designed to interoperate with an OpenStack deployment. In particular, our implementation heavily relies on, and extends, the monitoring capabilities offered by Monasca. Figure 3 shows how our predictive component enhances the typical elasticity control loop of a horizontally-scalable set of Nova compute instances (VMs). In our implementation, the actual resource orchestration is performed by Senlin.

Our approach works as follows. As soon as new system-level measurements from the compute instances are ingested and made available through the Monasca API, we leverage on time-series forecasting techniques to predict the future dynamics of some relevant infrastructure metrics. These are assumed to be key indicators for the amount of load that the system is handling (e.g., CPU utilization). Notice that, depending on the chosen time-series forecasting algorithm, it might be necessary to train the resulting model on a large portion of historical monitoring data. In addition, given the high velocity at which the operating conditions change, as typical in cloud environments, such a model most likely has to be updated frequently to overcome the effects of concept drift and keep getting accurate forecasts. At the moment, these aspects are assumed to be handled offline, such that the forecasting component must be provided with a pre-trained version of the chosen model that is ready to be used for inference.

The predictor is fed with a time-series in input that is obtained by aggregating the time-series of the target metric as measured from the VMs composing the elasticity group. The forecasts output by the predictor are then persisted back to Monasca, such that other infrastructure components can consume them. In addition, these predicted metrics become available to operators through standard monitoring dashboards (such as Grafana). In our case, the predicted metrics are used to feed a *threshold-based* scaling policy. This type of policies typically triggers a scale-out/scale-in action as soon as the specified upper/lower threshold is repeatedly reached for a certain number of subsequent observations. In our implementation, the threshold checks are performed by the highly-scalable alerting pipeline provided by Monasca through Apache Storm. Then, depending on the conditions of the system, Monasca notifies the Senlin orchestration engine regarding the scaling actions to be performed.

As detailed in the next section, in our experimentation we considered as predictors: a linear regressor, an MLP, and an RNN (see Section 3.6).

5 EXPERIMENTS

In this section, we report experimental results showing the performance of the approach described in Section 4, compared to a classical *reactive* threshold-based scaling strategy.

5.1 Experimental Set-up

From an infrastructural standpoint, as explained in details in Section 4, our test application leveraged on: *(i)* Senlin to orchestrate a horizontally-scalable cluster of Nova instances; *(ii)* Octavia to provide the cluster with load-balancing capabilities; *(iii)* Monasca to ingest the system-level metrics and to trigger the scaling actions; *(iv)* the forecasting component, developed by us, to estimate the future state of the system and to enable the *proactive* auto-scaling strategy. The Senlin cluster was configured to have a minimum of 2 active Nova instances and to expand up to 5. Each instance was configured to run Ubuntu 20.04 cloud image and to have 1 vCPU and 2 GB of RAM available. The Octavia load-balancer was configured to distribute the traffic among the active instances according to a simple *round-robin* strategy. Monasca was configured such that new CPU usage measurements were collected each minute. The forecasting component was configured to output a new prediction with the same interval, using the last 20 minutes worth of data as input. In particular, the input to the underlying forecasting model consisted in a time-series reporting the sum of the CPU usage measurements of the currently active instances. The output of the model was the estimated value in 15 minutes of the same time-series. The output was then divided by the number of currently active instances to get an estimate of the *average CPU usage* in 15 minutes (assuming the cluster size to be constant) and then persisted back to Monasca. We used PyTorch to implement standard flavors of MLP and RNN (see Section 3.6), while we used the implementation of the linear regressor provided by Scikit-learn.

To implement the *predictive* scaling strategy, the alerting component of Monasca was configured to trigger a scale-out action whenever the *predicted* average CPU usage of the cluster, as outputted by the forecasting component, reached the 80% for 3 times in a row. On the other hand, to implement the classical *reactive* scaling strategy, the alerting component was set to track the *actual* average CPU usage of the cluster in a similar fashion. In both cases, the alerting component was set to trigger a scale-in action whenever the *actual* average CPU usage of the cluster reached the 15% for 3 times in a row. Notice that such settings impose a delay of at least 3 minutes for an action to be triggered. In addition, each scaling action could adjust the size of the cluster by 1 instance only and could only take effect if it was triggered after at least 20 minutes since the last effective action (i.e., *cooldown* period). After a scale-out action was triggered and the resulting new instance was spawned, we imposed an additional delay of 6 minutes before such instance could start serving requests, emulating what could happen in a real production service deployment, where spinning up a new VM hosting complex software might take a non-negligible amount of time.

During our test runs, we generated traffic on the system using *distwalk* [9], an open-source distributed processing emulation tool developed by us. Such tool consists in a server module, that is started at boot on each instance and waits for TCP requests, and a client module, that sends, in this case,

requests to the load-balancer with the aim of increasing the CPU utilization of the instances. The client was configured such that it spawned 6 threads, each one provided with a ~1.5h-long trace reporting the operation rates (i.e., requests per second) that should be maintained for an interval of one minute each. Each thread was also forced to break its work in 1000 sessions, such that a new session opened a new connection with the load-balancer, that in turn selected a (possibly different) target instance.

The experiments were carried out using an all-in-one deployment of OpenStack (*Victoria* release), that was deployed using the tools provided by [29]. Notice that, in this case, the various OpenStack services are installed and operated within several Docker containers, not on bare metal. The deployment was hosted on a Dell R630 dual-socket test-bed, equipped with: 2 Intel Xeon E5-2640 v4 CPUs (2.40 GHz, 20 virtual cores each); 64 GB of RAM; Ubuntu 20.04.2 LTS operating system.

The implementation of the forecasting component was written in Python and is released under an open-source license [16]. For the sake of reproducibility, this work comes with a publicly available companion repository [17] including: the Heat templates used to set up the infrastructure; all the configuration files for the tools we used for our experiments; the synthetic dataset used to train the neural models and their pre-trained versions.

5.2 Results

Figure 4 reports the obtained results from running the described experiment, using 4 different scaling strategies, in presence of a workload—similar to the one depicted in Figure 1—that suddenly ramps up in a window of about 30 minutes (from time 50 to time 80 on the x-axis in the plots). In the CPU usage plots (left-hand side), the blue curve represents the ideal workload to be exercised on the whole cluster as input to each thread of the distwalk client, which submits its requests to the Senlin cluster through the Octavia load-balancer. This results in the actual per-VM workload highlighted in the first 3 curves in each plot. Additionally, the red curve in Figures 4c, 4e and 4g highlights the predicted average CPU utilization for the whole cluster, assuming the size of the cluster to remain constant, according to the different adopted predictive policies.

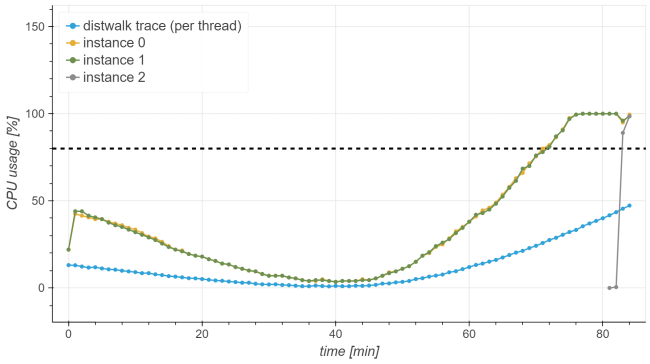
As visible, the standard threshold-based scaling strategy tuned on the average CPU utilization crossing the 80% threshold, shown in Figure 4a, fails to scale-up the cluster on time, as we need 3 consecutive violations of the threshold in order to start the scale-out operation. Additionally, the start-time of the new VM takes as much as 6 minutes, therefore the scale-out decision happens approximately at minute 73, albeit the new VM starts serving requests only approximately at minute 83, after the cluster has been overloaded for about 8 minutes at full CPU utilization. Indeed, observing the obtained client-side response times in Figure 4b, we can see that this resulted in heavy degradation of the performance. The other 3 plots correspond to the use of a predictive policy, with 3 different predictors. The simple LR-based prediction policy shown in Figure 4c, performs a better job at predicting the growth of the CPU utilization on time. Indeed, it activates the scale-out earlier, approximately at minute 65. The anticipated scaling has a clear effect on the observed client-side response times (Figure 4d), which stay all below 25 ms. In Figures 4e and 4g, we can see the effects of using a simple MLP and RNN, respectively, as predictors. In these cases, we can make similar observations about the benefits of the anticipated scaling, as in the LR-based case. However, with respect to the latter one, the neural networks-based approaches manage to capture the non-linear behavior of the input ramp, which exhibits a clear upwards curvature. Therefore, they scale even slightly earlier, approximately at minutes 60. For instance, the MLP-based policy results in slightly better response times in the worst case during the considered time-frame. However, overall, all considered predictive solutions perform similarly from the view-point of the client-side performance. For completeness, Table 1 reports the average and various percentile values of the response-times observed by the client during the time window shown in the previous pictures. Additionally, Table 2 reports the overhead imposed by the proposed forecasting component at each activation that, in our experiments, happened once per minute. Notice that the *total* time includes interacting with Monasca APIs to fetch input data. Notice that in all experiments, even though the scaling policies expand the cluster, reducing the per-VM workload, the input trace keeps growing, requiring additional scale-out actions, which are inhibited due to our 20 minutes cooldown period. This is among the engineering issues to be addressed in future extensions of the proposed architecture.

Table 1: Descriptive statistics of the client-side response times observed during the experiments.

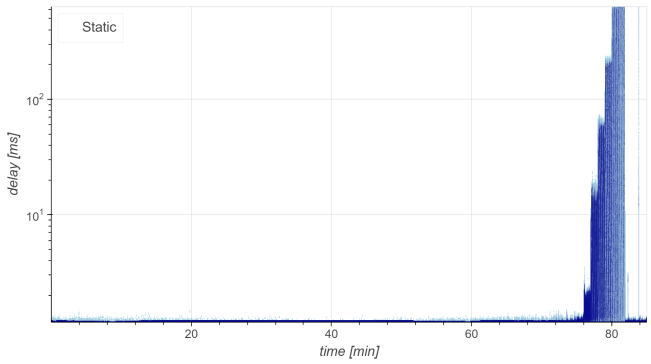
	Static	LR	MLP	RNN
avg (ms)	43.67	2.02	2.07	2.16
p90 (ms)	166.75	2.70	2.84	2.76
p95 (ms)	329.25	3.43	3.77	3.61
p99 (ms)	577.29	3.90	4.07	4.01
p99.5 (ms)	633.81	4.23	4.20	4.12
p99.9 (ms)	731.63	8.36	7.17	9.97

Table 2: Average overhead imposed by the proposed forecasting component.

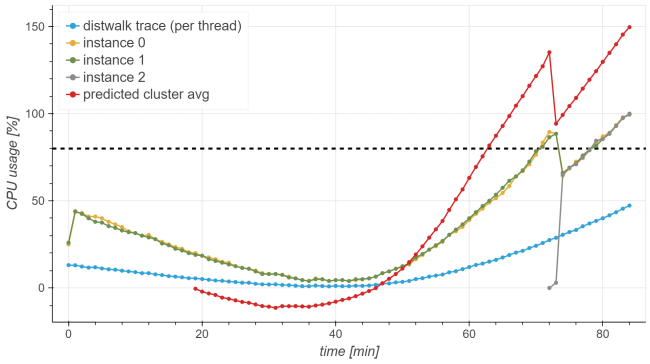
	LR	MLP	RNN
forecasting time (ms)	64.89	61.12	77.85
total time (ms)	285.34	224.66	308.13



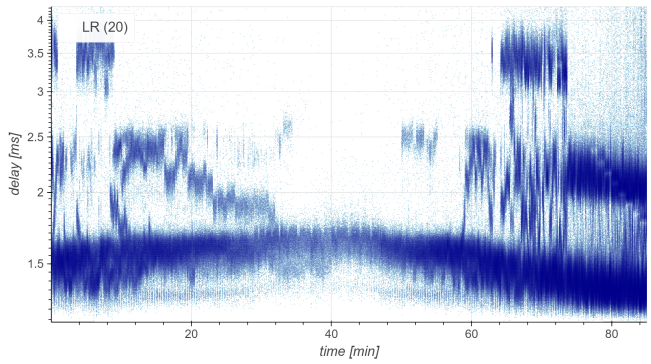
(a) Static scaling policy - CPU usage



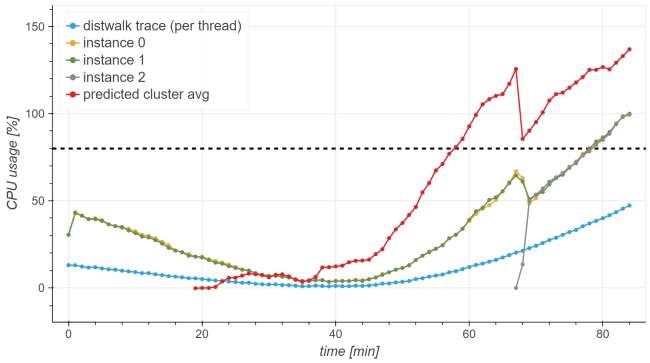
(b) Static scaling policy - client-side response time



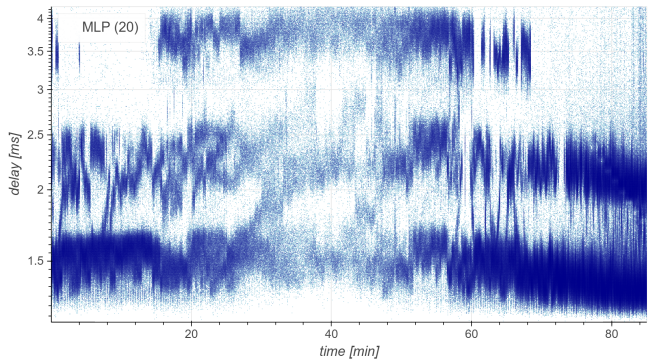
(c) LR-based scaling policy - CPU usage



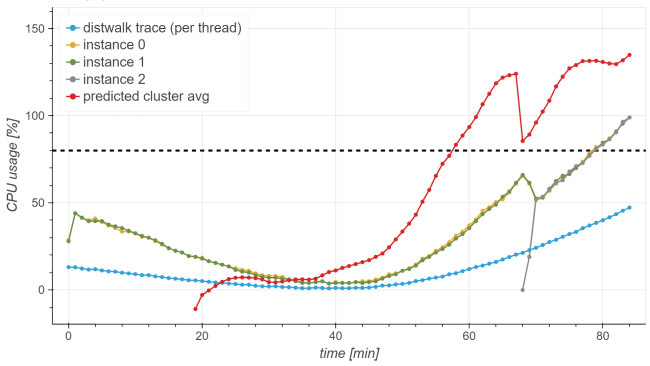
(d) LR-based scaling policy - client-side response time



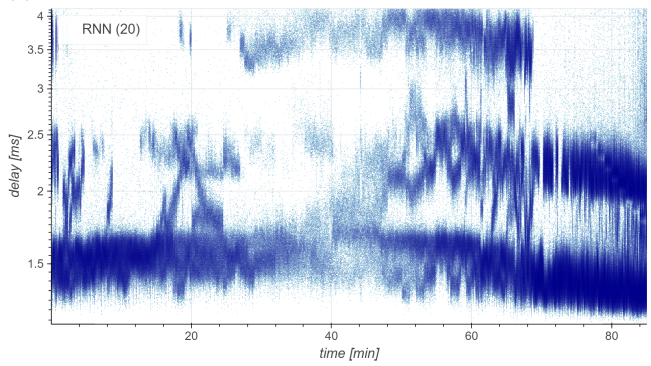
(e) MLP-based scaling policy - CPU usage



(f) MLP-based scaling policy - client-side response time



(g) RNN-based scaling policy - CPU usage



(h) RNN-based scaling policy - client-side response time

Figure 4: Experimental results.

6 CONCLUSIONS AND FUTURE WORK

In this paper, an architecture for predictive elasticity in cloud infrastructures has been presented. The approach, prototyped in OpenStack leveraging on the Monasca API, allows for the automatic creation of predictive metrics that are continuously updated reflecting the expected evolution of the system state in a near future. These metrics can be seamlessly combined with the regular instantaneous ones already available through the standard OpenStack monitoring infrastructure, to build cluster operation and management policies that go beyond purely reactive strategies. As a case-study, we presented an experimentation exploiting the presented architecture for realizing a predictive elasticity controller via Senlin, that applies autoscaling decisions trying to anticipate possible workload surges that might not easily be handled through classical threshold-based autoscaling rules. The proposed approach is particularly useful for services with non-negligible instance spawning times, as commonplace in production services in which creating new VMs may require tens of minutes.

In the future, we plan to integrate our metric forecasting component better within the OpenStack eco-system, introducing a number of standard metric forecasting predictors, either univariate and multi-variate, which can easily be deployed through the OpenStack command-line interface.

Regarding the orchestration logics, we plan to apply additional AI techniques to experiment with alternate scaling policies (rather than threshold-based ones), like those based on reinforcement learning, for example.

Regarding the crucial problem of training and re-training the model, we plan to provide a means for periodic re-training of the model based on updated data already available in Monasca, but we also plan to introduce a method for detection of concept drift, triggering a re-training of the model whenever the one in use starts exhibiting a drop in accuracy.

Finally, we plan to validate at a deeper level our architecture by considering data sets from real production workloads, and deal with scalability issues due to the need for considering massive deployments where one might have several predictive elasticity loops to act on several services. In this regard, we plan to leverage on the scalable analytics processing architecture provided by Monasca through its integrated Storm component.

REFERENCES

[1] Hamid Arabnejad, Claus Pahl, Pooyan Jamshidi, and Giovani Estrada. 2017. A Comparison of Reinforcement Learning Techniques for Fuzzy Cloud Auto-Scaling. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 64–73.

[2] Carlos Hernán Tobar Arteaga, Fulvio Rissoi, and Oscar Mauricio Caicedo Rendon. 2017. An adaptive scaling mechanism for managing performance variations in network functions virtualization: A case study in an nfv-based epc. In *13th International Conference on Network and Service Management*. 1–7.

[3] Davide Bacciu, Federico Errica, Alessio Micheli, and Marco Podda. 2020. A gentle introduction to deep learning for graphs. *Neural Networks* 129 (2020).

[4] Abul Bashar. 2013. Autonomic scaling of Cloud Computing resources using BN-based prediction models. In *2013 IEEE 2nd International Conference on Cloud Networking (CloudNet)*. 200–204.

[5] Michael Borkowski, Stefan Schulte, and Christoph Hochreiner. 2016. Predicting Cloud Resource Utilization. In *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*. 37–42.

[6] Sandro Brunner, Martin Blöchlinger, Giovanni Toffetti, Josef Spillner, and Thomas Michael Bohnert. 2015. Experimental Evaluation of the Cloud-Native Application Design. In *Proceedings of the 8th International Conference on Utility and Cloud Computing* (Limassol, Cyprus) *(UCC '15)*. IEEE Press, 488493.

[7] Rajkumar Buyya, James Broberg, and Andrzej M. Goscinski. 2011. *Cloud Computing Principles and Paradigms*. Wiley Publishing.

[8] Giuseppe Antonio Carella, Michael Pauls, Lars Grebe, and Thomas Magedanz. 2016. An extensible autoscaling engine (ae) for software-based network functions. In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 219–225.

[9] Tommaso Cucinotta. 2021. distwalk. https://github.com/tomcucinotta/distwalk

[10] Tommaso Cucinotta, Giacomo Lanciano, Antonio Ritacco, Fabio Brau, Filippo Galli, Vincenzo Iannino, Marco Vannucci, Antonino Artale, Joao Barata, and Enrica Sposato. 2021. Forecasting Operation Metrics for Virtualized Network Functions. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 596–605.

[11] Xincai Fei, Fangming Liu, Hong Xu, and Hai Jin. 2018. Adaptive VNF Scaling and Flow Routing with Proactive Demand Prediction. In *IEEE Conference on Computer Communications*, Vol. 2018-April. 486–494.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[13] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. 2012. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems* 28, 1 (2012), 155–162.

[14] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre Alain Muller. 2019. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery* 33, 4 (2019), 917–963.

[15] Peng Kang and Palden Lama. 2020. Robust Resource Scaling of Containerized Microservices with Probabilistic Machine learning. In *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. 122–131.

[16] Giacomo Lanciano. 2021. monasca-predictor. https://github.com/giacomolanciano/monasca-predictor

[17] Giacomo Lanciano and Filippo Galli. 2021. predictive-auto-scaling-openstack. https://github.com/giacomolanciano/UCC2021-predictive-auto-scaling-openstack

[18] Nikolay Laptev, Jason Yosinski, Li Erran Li, and Slawek Smyl. 2017. Time-series Extreme Event Forecasting with Neural Networks at Uber. *International Conference on Machine Learning - Time Series Workshop* (2017), 1–5.

[19] C. Makaya, D. Freimuth, D. Wood, and S. Calo. 2015. Policy-based NFV management and orchestration. In *IEEE Conference on Network Function Virtualization and Software Defined Network*. 128–134.

[20] Pankaj Malhotra, TV Vishnu, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. 2017. TimeNet: Pre-trained deep recurrent neural network for time series classification. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*.

[21] Peter Mell and Tim Grance. 2011. The NIST Definition of Cloud Computing. SP 800–145. https://csrc.nist.gov/publications/detail/sp/800-145/final.

[22] R. Mijumbi, S. Hasija, S. Davy, A. Davy, B. Jennings, and R. Boutaba. 2017. Topology-Aware Prediction of Virtual Network Function Resource Requirements. *IEEE Transactions on Network and Service Management* 14, 1 (2017), 106–120.

[23] Masanori Miyazawa, Michiaki Hayashi, and Rolf Stadler. 2015. vNMF: Distributed fault detection using clustering approach for network function virtualization. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 640–645.

[24] NFV Industry Specif. Group. 2012. Network Functions Virtualisation. Introductory White Paper. http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[25] Carlos H.Z. Nicodemus, Cristina Boeres, and Vinod E.F. Rebello. 2020. Managing Vertical Memory Elasticity in Containers. In *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. 132–142.
[26] Tomonobu Niwa, Masanori Miyazawa, Michiaki Hayashi, and Rolf Stadler. 2015. Universal fault detection for NFV using SOM-based clustering. In *17th Asia-Pacific Network Operations and Management Symposium*. 315–320.
[27] OpenStack. 2021. Cinder Documentation. https://docs.openstack.org/cinder
[28] OpenStack. 2021. Glance Documentation. https://docs.openstack.org/glance
[29] OpenStack. 2021. Kolla Documentation. https://docs.openstack.org/kolla
[30] OpenStack. 2021. Monasca Documentation. https://docs.openstack.org/monasca
[31] OpenStack. 2021. Neutron Documentation. https://docs.openstack.org/neutron
[32] OpenStack. 2021. Nova Documentation. https://docs.openstack.org/nova
[33] OpenStack. 2021. Octavia Documentation. https://docs.openstack.org/octavia
[34] OpenStack. 2021. Senlin Documentation. https://docs.openstack.org/senlin
[35] Sabidur Rahman, Tanjila Ahmed, Minh Huynh, Massimo Tornatore, and Biswanath Mukherjee. 2018. Auto-scaling VNFs using machine learning to improve QoS and reduce cost. In *IEEE International Conference on Communications*.
[36] Syama Sundar Rangapuram, Matthias Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. 2018. Deep state space models for time series forecasting. In *Advances in Neural Information Processing Systems*, Vol. 2018-Decem. 7785–7794.
[37] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. 2011. Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting. In *2011 IEEE 4th International Conference on Cloud Computing*. 500–507.
[38] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. 3104–3112.
[39] Pengcheng Tang, Fei Li, Wei Zhou, Weihua Hu, and Li Yang. 2015. Efficient auto-scaling approach in the telco cloud using self-learning algorithm. In *2015 IEEE Global Communications Conference (GLOBECOM)*. 1–6.
[40] TPC. 2021. TPC-W Benchmark. http://www.tpc.org/tpcw/
[41] Duc Truong and Jude Cross. 2019. How Blizzard Entertainment Uses Autoscaling With Overwatch. https://www.openstack.org/videos/summits/denver-2019/how-blizzard-entertainment-uses-autoscaling-with-overwatch
[42] Yikai Xiao, Qixia Zhang, Fangming Liu, Jia Wang, Miao Zhao, Zhongxing Zhang, and Jiaxing Zhang. 2019. NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning. In *Proceedings of the International Symposium on Quality of Service*, Vol. 19. 1–10.
[43] Zakia Zaman, Sabidur Rahman, and Mahmuda Naznin. 2019. Novel Approaches for VNF Requirement Prediction Using DNN and LSTM. In *2019 IEEE Global Communications Conference (GLOBECOM)*. 1–6.