ENVI: Elastic resource flexing for Network function VIrtualization

Lianjie Cao   Puneet Sharma   Sonia Fahmy   Vinay Saxena
Hewlett Packard Labs   *Purdue University*   *Hewlett Packard Enterprise*

Abstract

Dynamic and elastic resource allocation to Virtual Network Functions (VNFs) in accordance with varying workloads is a must for realizing promised reductions in capital and operational expenses in Network Functions Virtualization (NFV). However, workload heterogeneity and complex relationships between resources allocated to a VNF and the resulting capacity makes elastic resource flexing a challenging task. We propose an NFV resource flexing system, ENVI, that uses a combination of VNF-level features and infrastructure-level features to construct a machine-learning-based decision engine for detecting resource flexing events. ENVI also extracts the dependence relationship among VNFs in deployed Service Function Chains (SFCs) to carefully plan the sequence of resource flexing steps upon scaling detection. We present preliminary results for the accuracy of ENVI's resource flexing decision engine with two different VNFs, namely, the caching proxy Squid and the intrusion detection system Suricata. Our preliminary results show that using a combination of features to train a neural network model is a promising approach for scaling detection.

1 Introduction

Motivated by the success of cloud computing, Communication Service Providers (CSPs) are adopting Network Functions Virtualization (NFV) to virtualize network functions (NFs) and deploy them on commodity compute, storage and networking resources. The goal of NFV is to bring agility to NF deployment and leverage elastic scaling to reduce overall operational expenses. *Elastic resource flexing* in the NFV ecosystem allows an orchestrator to provision appropriate resources to a particular VNF for matching workload dynamics, either by increasing (or reducing) the resource allocations of already deployed VNF instances or increasing (or reducing) the number of VNF instances. This paper considers *elastic resource flexing* mechanisms for VNF management.

NFV deployments comprise sets of VNF instances implementing network services (*e.g.*, intrusion detection systems (IDSes), load balancers, caching proxies) hosted on private or public cloud platforms such as virtual machines (VMs) or containers. It is common to connect VNF instances in a particular order that network traffic needs to traverse, referred to as a Service Function Chain (SFC). Virtualization allows each VNF to be elastically scaled to use more (or less) virtualized resources *on demand*. We observe that resource consumption-based thresholds for detecting overload in standard cloud computing environments are insufficient for NFV deployments. Accuracy and timeliness of scaling detection allow balancing the tradeoffs associated with VNF resource allocation. Detecting scaling long before actual overload causes under-utilization of resources allocated to a VNF (and hence higher operational expenses). Alternatively, detection after the fact can incur penalties associated with service disruption.

Several challenges complicate the design of effective VNF scaling detection mechanisms. First, unlike hardware implementations, VNF vendors may not provide detailed capacity/performance specifications of VNFs. This is because system performance/capacity of VNFs depends on underlying NFV infrastructure, resource sizing and workload dynamics [5] which makes it challenging for VNF vendors to provide complete performance/capacity information. For instance, the maximum throughput of an IDS running in a VM is related to the configuration of the physical server, *e.g.*, a vCPU mapped to an Intel Xeon processor offers different traffic processing power from an AMD Opteron processor. Second, each VNF has a distinct processing logic depending on incident network traffic and events. Even if certain VNFs share packet processing functionality such as packet header analysis, the differences in upper-layer processing and implementation can exhibit unique resource usage patterns. Third, the dynamics of network traffic in volume and composition may trigger different processing units of the same VNF (*e.g.*, rule match in IDS) and hence consume different amounts of resources. As an example, Table 1 shows different capacity values (number of completed HTTP requests per second) and CPU usage on our testbed for Squid, a caching proxy NF, when varying HTTP response size from 10 KB to 100 KB. Fourth, unlike similar scenarios (*e.g.,* cloud applications), network traffic (constituting the VNF workload) is forwarded through VNFs based on the SFC forwarding graph.

Table 1: Squid capacity for different HTTP response sizes

Response Size	10 KB	50 KB	90 KB
Capacity (requests/sec)	2767	1232	723
CPU Usage (%)	74	51	54

In this paper, we describe ENVI, a modular VNF resource management system that periodically collects VNF-specific and infrastructure resource utilization information, and detects VNF scaling using this information and pre-trained machine learning models. ENVI generates resource flexing plans considering SFC relationships. ENVI makes three novel contributions:

(i) Composition of VNF-specific information and infrastructure resource usage information for detection of VNF scaling. Most VNFs (actively or passively) report internal statistics to administrators for debugging, monitoring and security. This internal information (such as request queue size) is critical to ensure optimal functioning. Although VNFs of the same type may differ in implementation, they usually use similar metrics. For instance, both Snort and Suricata, as IDSes, report packet classification, throughput, and rule matching statistics. We argue that combining VNF-specific information with resource utilization information (referred to as VNF-level features and infrastructure-level features, respectively) can enhance our understanding of VNF dynamics at runtime, and hence yield more accurate scaling detection.

(ii) Modeling scaling detection as a classification problem. Most systems today use a static policy-based mechanism on one or multiple resource usage statistics to detect overload. This approach fails to consider the impact of the VNF processing logic, implementation details, and traffic dynamics. While it is almost impossible to establish precise mathematical relationships among all factors, it is feasible to train machine learning models to detect scaling *online*. ENVI uses machine learning models to capture sophisticated relationships between VNF runtime status and system scaling.

(iii) Resource flexing planning based on service chain relationships. ENVI's resource provisioning system not only captures the VNF status, workload and infrastructure information in real-time, but also considers the chaining relationship between VNF instance sets. This consideration allows concerted resource flexing across all components of a service function chain, thus reducing transition times and service disruptions.

In this paper, we demonstrate the use of VNF-level and infrastructure-level information as input features to train machine learning models for scaling detection. We report preliminary experimental results on the caching proxy Squid and the IDS Suricata.

2 Related Work

Previous work (*e.g.,*, [21, 8, 18, 28, 20, 7, 13, 32, 16]) considered VNF resource allocation together with VNF placement, and modeled them as an optimization problem (*e.g.*, Integer Linear Programing (ILP)) that minimizes the total number of VNF instances, communication cost, or deployment cost under constraints of network traffic and physical network topology. This line of work assumes VNF capacities are known and static, ignoring the performance characterization challenges. The work does not consider the impact of dynamic network traffic on different types of VNFs. Hence, the resource allocation solutions may not work well in real-world deployments.

Stratos [12] and E2 [25] propose comprehensive NFV orchestration frameworks which manage VNF instances and distribute flows efficiently. However, E2 [25] relies on the VNF developer to give the overload indicator for scaling a VNF with more instances. The resource provisioning strategy in Stratos [12] is the closest to our solution. However, Stratos monitors OS-level statistics only and does not specify how they are used to detect overload.

In the cloud computing domain, resource allocation is an important problem. However, on the public cloud, neither the provider nor the user is willing to share information, which limits the ability to make efficient resource allocation decisions. In addition, the strict chaining requirements in NFV introduces special constraints. Most cloud platforms and third-party developers simply provide policy-based interfaces (*e.g.*, OpenStack Heat [24], Amazon AWS AutoScaling [2], Google Compute Engine AutoScaler [15] and RightScale [26]) to users to scale their application by monitoring basic infrastructure-level information. Some research work (*e.g.*, [14], [27] and [31]) assumes certain workload patterns exist, and identifies and stores resource assignment solutions for future use.

Learning approaches have proven effective in solving system problems. Nagaraj et al. [22], Liu et al. [17] and Arzani et al. [1] apply machine learning techniques to investigate system/network anomalies. Nagaraj et al. [22] compare distributed system logs using statistical tests and dependency networks to identify performance degradation. Liu et al. [17] and Arzani et al. [1] train random forest models on historical network data to determine network anomalies and root causes of failures, respectively. Gao et al. [11] predict power usage effectiveness (PUE) in Google data centers. Bao et al [3] guide cellular network resource allocation using user experience prediction. Mao et al. [19] solve the multi-resource allocation problem in a reinforcement learning framework. These approaches are orthogonal to our work.

3 ENVI Design

ENVI introduces three novel mechanisms to VNF resource management: (i) combination of VNF-specific information (VNF-level features) and system resource utilization (infrastructure-level features) for a deeper understanding of VNF runtime dynamics, (ii) application of machine learning to detect VNF scaling, and (iii) resource adjustment leveraging chaining relations.

As shown in Fig. 1, we design ENVI with four decoupled modules leveraging the NFV architecture from ETSI [10]: **VNF monitor**, **scaling decision engine**, **resource flexing engine** and **placement engine**. We intentionally decouple these building blocks to test different algorithms and reuse existing solutions. In this section, we explain each module and how it interacts with other modules.

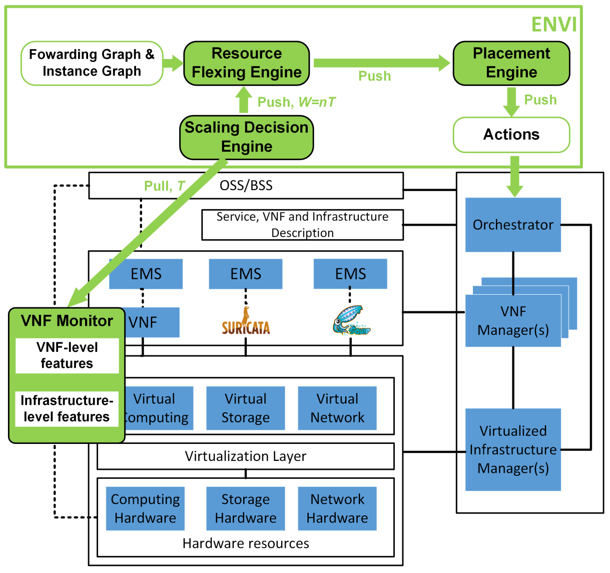


Figure 1: ENVI Architecture.

3.1 Detecting Scaling

Prior work (*e.g.*, [6, 34]) reports improved performance when leveraging application information in scheduling. Such information, however, is not always available due to privacy concerns. Fortunately, in NFV, the infrastructure and VNFs are typically owned by the same administrator who has access to both. In ENVI, a VNF monitor queries each VNF instance to collect VNF-level and infrastructure-level feature information using two separate lightweight monitoring threads every time interval T. ENVI uses a distinct monitoring agent for each VNF. Most VNFs report key metrics through log files or APIs which are inexpensive to query. The VNF monitor converts and stores all metric values in a key-value format as VNF-level feature time-series information.

Our goal is to determine the appropriate time when a VNF needs to be scaled and allocate additional resources to it at that time. If we are too late, the VNF gets overloaded, leading to service quality violations. If we are too early, we allocate unnecessary resources which causes inefficient resource usage. To this end, we must formulate the relation among VNF runtime status, resource utilization, and scaling decision. We find that it is infeasible to formulate exact mathematical models that consider the inherent heterogeneity in VNF functionality and implementation and the physical infrastructure. We propose to use machine learning models for our scaling decision engine (SDE) to learn how to make scaling decisions based on data collected from offline performance tests, and then use and update the learned models in our operational environment.

The SDE module pulls the values of VNF-level and infrastructure-level features from the VNF monitor every time window $W = nT$ to avoid overreaction (n is set to 10 by default) and converts these values to numerical values (if needed). The module then computes statistical measures (*e.g.*, max, min, mean, median and variance) over W to capture temporal dynamics and use these values as a data point. SDE includes two phases: an *offline training phase* and an *online operation phase*.

During the offline training phase, we conduct a series of training experiments covering as many types of workloads as possible using our VNF performance testing framework [5]. We label each data point for time window W in the collected training data with a 0 or 1 meaning "do not scale" and "scale," respectively. Extending the methodology in our previous work [5], our tests determine the maximum capacity C_i for VNF i. We label a data point 1 if the input workload rate $R_{ij} > \alpha C_i, 0 < \alpha \leq 1$ during the jth time window T_j, which means that we attempt to scale the VNF before reaching its maximum capacity. The parameter α controls the time to enforce the scaling decision and appropriate resource allocation.

Note that the labeling process can be VNF-specific, since users can define service quality violations for different VNFs differently. We then use the collected data to train an initial model. We evaluate our initial model on different workload types in Section 4.

The initial model is used to predict scaling events during the online operation phase. To cope with workload variations that were not captured during offline training, ENVI continues collecting and labeling new operational data and updating the initial model in a background process using online learning algorithms. In some cases, the SDE may be late in generating "scale" events due to model inaccuracies. To address this contingency, we allow the SDE to generate a third type of event, "urgent scale" (with value 2), if VNF failure or severe overload is detected. This event is pushed to the resource flexing engine immediately, and the SDE performs a model update immediately as well.

3.2 Neural Network Model

We explored multiple machine learning models (*e.g.*, decision tree, random forest, logistic regression, Lasso and naïve Bayes) for the SDE. we selected a fully connected *neural network* model with four layers: input layer, two hidden layers and output layer. We select this approach because: (1) a neural network is able to construct new features through customizable hidden layers and fit nonlinear functions when an explicit mathematical formulation is unavailable, and (2) the neural network can model dependence of input features and data points, which is important for VNF-level features and VNF operational data. For instance, for Suricata, VNF-level feature `decoder.bytes` is related to `decoder.pkts`, `decoder.max_pkt_size` and `decoder.avg_pkt_size`. Our experiments show that the neural network model is able to effectively capture VNF scaling behavior if we label data points appropriately, and VNFs report relevant features. However, interpreting a trained neural network and feature importance remain open problems.

Machine learning models need retraining when the underlying hardware used for initial training changes. Fig. 2 shows the cross-test of initial neural network models on different hardware configurations. The results demonstrate that 90% to 95% of the accuracy is achieved on the same hardware configuration by standardizing the values of input features with composite features. The initial models can be improved during operation. However, the initial models may not perform well if VNFs undergo major software updates that affect their capacity and performance. For example, we have observed that Suricata, starting with version v3.1, significantly improved its pattern matching performance. In such cases, retraining the initial neural network model is important.

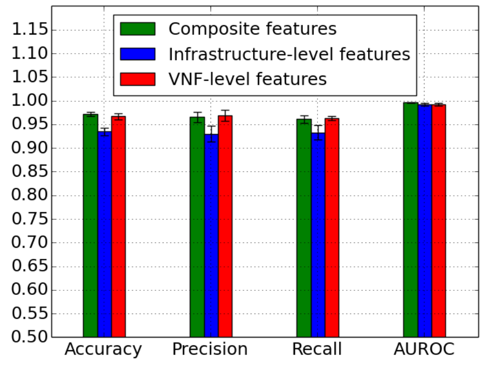


Figure 2: Cross-testing initial neural network models (using composite, infrastructure-level and VNF-level features) for Suricata on different hardware configurations. Each bar represents normalized values over the same measure derived from 5-fold cross-validation.

During the *online operation phase*, the SDE updates the initial neural network models periodically based on newly collected data. We use the previous values as initial weights to train the model on new data. If the workload shows significant difference, we increase the weight of new data to keep up with workload variations.

3.3 Flexing Resources

When "scale" or "urgent scale" events are generated, the SDE pushes a *scale array* indicating scaling requirements for each instance to the resource flexing engine (RFE), along with related VNF-level and infrastructure-level feature information. We extend the strategy in Stratos [12] to a "multi-stage resource adjustment" strategy. On receiving the scale array from the SDE, the RFE first scales VNF instances that are labeled "urgent scale." It then reduces the *multi-VNF adjustment* (*e.g.*, more than one VNF needs to be scaled) to multiple *single-VNF adjustments* by leveraging the chaining relation defined by the SFC policy graph. For each single-VNF adjustment, the RFE first attempts to adjust the VNF status by redistributing the workload among all instances of a VNF. If this is possible, we redistribute flows by reconfiguring traffic forwarding rules. Otherwise, the RFE scales out/up the affected VNF based on the scaling preferences obtained during the offline phase.

Once the resource flexing plan is generated, the RFE pushes it to the placement engine (PE). The PE converts the received plan to executable and platform-dependent commands (*e.g.*, OpenStack Heat templates) and sends these commands to the NFV orchestrator to enforce. This stage involves optimization to determine how to place new VNF instances to reduce costs and processing overhead. PE details are beyond the scope of this work: users can plug in existing solutions (*e.g.*, [21, 8, 4, 18, 28, 20, 7, 13, 32, 16]).

4 Prototype Evaluation

We implemented a prototype of ENVI and evaluated it on a testbed with three HP DL360p blade servers and two HP Z420 workstations, connected by an HPE 3500yl Gigabit switch (for the management network) and an HPE 5820X 10 Gigabit switch (for the instance network). All the compute and network resources are managed by OpenStack Kilo [23].

Due to the lack of public traces for VNF benchmarking, we synthetically generate the workload by extending the methodology used in NFV-VITAL [5] to evaluate ENVI with two VNFs: caching proxy Squid [29] (version 3.3.8) and IDS software Suricata [30] (version 3.2.1). Each VNF is tested with ten different workloads: different HTTP response sizes (10 to 100 KB) for Squid and different malicious traffic fractions (0 to 90%) for Suricata. For each workload, we first gradually increase the workload generation rate until the VNF reaches the maximum system capacity where $\frac{System\ Throughput}{Input\ Workload} \geq 99\%$. Then, we randomly pick a workload generation rate that is smaller than the maximum system capacity to collect data to train and test the SDE. We do not overload the VNF frequently in our experiments, since the goal of the SDE is to generate "scale" notifications prior to VNF overload. We collected over 1000 data points over 10 workload types for each VNF in our experiments.

Ideally, we want to train an initial model for each VNF using offline experimental data and gradually improve it online as more workload patterns are observed. In this section, we evaluate the performance of a neural network model that is trained and tested on disjoint workload types. More specifically, we train the neural network model on n workload types and test it on the remaining $10-n$ workload types, where $n = 1, 2, \cdots, 9$. For each n, we enumerate all possible combinations of workload types and run 5-fold cross-validation on selected data sets to avoid overfitting before testing on the remaining $10-n$ workload types. We compare the performance of the neural network (NN) with three other machine learning (ML) classification models: decision tree (DT), random forest (RF) and logistic regression (LR), using different performance measures: accuracy, precision, recall, receiver operating characteristic (ROC) and area under ROC curve (AUROC). We also compare with a baseline method that uses a scaling policy based on a CPU usage threshold. A similar training and evaluation methodology is used for other classification models.

4.1 Suricata Results

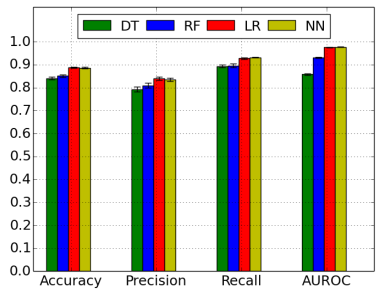
We configure Suricata with the latest Emerging rule set [9] and generate UDP traffic as benign traffic. For malicious traffic, we target a single rule which matches UDP port number and payload content. Suricata reports 100 quantities in its log file which are used as VNF-level features. Our resource monitor collects 8 basic resource utilization metrics as infrastructure-level features.

Fig. 3 compares the performance of different classification models trained with three different feature sets: VNF-level only, infrastructure-level only, and composite features. Compared to VNF-level features, infrastructure-level features yield up to 6% higher accuracy, 8% higher precision, 3% higher recall, and 6% higher AUROC. For Suricata, infrastructure-level features are generally better indicators for detecting "scale" events. This is because Suricata is a simple VNF that captures packets and generates alerts based on rule sets. The performance of Suricata is highly correlated to CPU utilization. This is consistent with the observed 72% accuracy of the baseline method, which only uses a CPU threshold to detect scaling. With composite features, all machine learning models give performance close to that with infrastructure-level features. The neural network model outperforms other classification models and consistently yields close to 90% accuracy even with only VNF-level features. The standard error for all measures is small, indicating that machine learning models are stable when trained on different numbers and types of workloads. This increases our confidence that the initial neural network model can offer a reasonable starting point for the online phase.

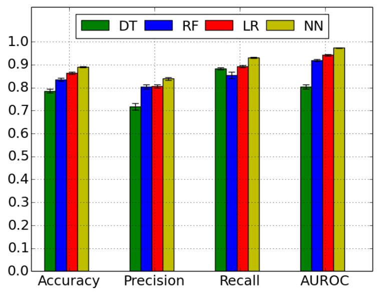
4.2 Squid Results

We use a caching proxy benchmarking tool, *Web Polygraph* [33], to generate HTTP requests from clients and simulate different types of workload by varying HTTP response sizes from 10 to 100 KB. We use *squidclient* (provided by Squid) to monitor the runtime status of Squid and collect 150 VNF-level features.

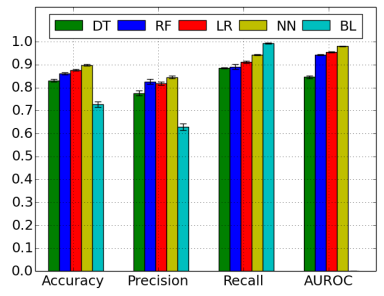
As shown in Fig. 4, infrastructure-level features yield significantly worse performance for all measures, unlike what we observed with Suricata. Squid is a more complex VNF than Suricata, and its performance is not highly correlated to CPU usage. To handle an HTTP request, Squid must accept a connection from a client, establish a connection with a server, and forward data from the server to the client. Therefore, infrastructure-level features are inadequate for accurate scaling detection. The baseline method also reports poor performance (44% accuracy). Performance with composite features and VNF-level features is better. The neural network



(a) Infrastructure-level features

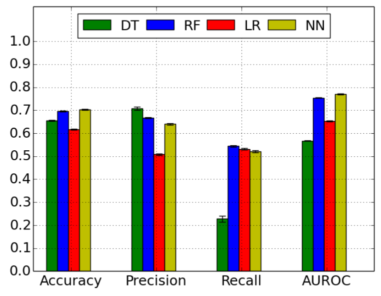


(b) VNF-level features

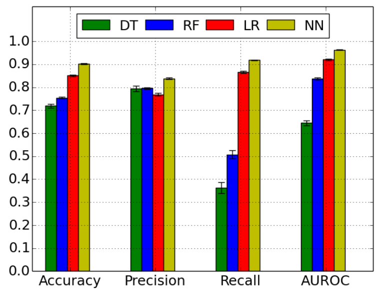


(c) Composite features

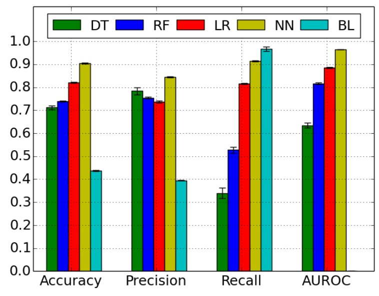
Figure 3: Statistical measures of Suricata tests.



(a) Infrastructure-level features



(b) VNF-level features



(c) Composite features

Figure 4: Statistical measures of Squid tests.

model again outperforms other models in all measures by 5%˜19% with small standard error. The baseline method reports a high recall value for both Suricata and Squid even when accuracy and precision are low: the baseline method generates more positive results than other models using our labeling mechanism. Reducing its CPU usage threshold increases its performance in some cases, but it still performs worse than other models, especially for complex VNFs like Squid.

5 Discussion

Model Feature Set. The performance of any machine learning (ML) model is limited by the availability of the right feature set for decision making. ENVI's decision engine will suffer if VNF vendors do not expose appropriate operational information. It is worth exploring the possibility of supplementing infrastructure-level information with OS-level information, such as memory allocation statistics, page fault information, device I/O information and system call information. Are there indirect mechanisms for extracting VNF specific features if they are not readily available?

Model Training Overhead. Our evaluation (Sec. 4) shows that the neural network model outperforms other classification models such decision tree, random forest and logistic regression. However, tuning a neural network model with the right set of features/parameters is a challenging task. Creating deep neural network models may also suffer from lack of training data and high training overhead. We believe that temporal correlation among features can be exploited for improved overload prediction. How should temporality be captured in ENVI's ML models? Could convolutional neural networks (CNN) or recurrent neural networks (RNN) improve the results?

Model Evolution. False negatives and false positives have different impacts on operation. False negatives mean that a VNF does not scale when it is should and result in service disruption. False positives lead to resource overprovisioning since a VNF unnecessarily scales. We need a utility/scoring function to capture the impact of both in terms of service quality and resource efficiency. While offline models are a good starting point, monitoring overload and failures in production environments can be used for online-training. What mechanisms are required for handling imbalanced labeling and using utility functions to bias model evolution during operation?

Finer-grained Resource Flexing. Currently the resource flexing engine performs coarse-grained search over a limited set of VM sizing options. More flexible resource allocation strategies are possible in highly customized or containerized deployments. Can reinforcement learning approaches, based on monitoring the impact of fine-grained resource allocation, benefit ENVI?

References

[1] B. Arzani, S. Ciraci, B. T. Loo, A. Schuster, and G. Outhred. Taking the blame game out of data centers operations with netpoirot. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 440–453. ACM, 2016.

[2] Amazon AWS - Auto Scaling. `https://aws.amazon.com/autoscaling/`.

[3] Y. Bao, H. Wu, and X. Liu. From prediction to action: A closed-loop approach for data-guided network resource allocation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1425–1434. ACM, 2016.

[4] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba. On orchestrating virtual network functions. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 50–56. IEEE, 2015.

[5] L. Cao, P. Sharma, S. Fahmy, and V. Saxena. NFV-Vital: A framework for characterizing the performance of virtual network functions. In *Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on*, pages 93–99. IEEE, 2015.

[6] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient coflow scheduling with varys. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 443–454. ACM, 2014.

[7] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca. The dynamic placement of virtual network functions. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9. IEEE, 2014.

[8] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz. Near optimal placement of virtual network functions. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 1346–1354. IEEE, 2015.

[9] Emerging Threats. `https://rules.emergingthreats.net/`.

[10] ETSI Network Functions Virtualisation (NFV) Architectural Framework. `http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf`.

[11] J. Gao and R. Jamidar. Machine learning applications for data center optimization. *Google White Paper*, 2014.

[12] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar. Stratos: A network-aware orchestration layer for middleboxes in the cloud. Technical report, Technical Report, 2013.

[13] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba. Elastic virtual network function placement. In *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, pages 255–260. IEEE, 2015.

[14] Z. Gong, X. Gu, and J. Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 9–16. Ieee, 2010.

[15] Google Compute Engine - AutoScaler. `https://cloud.google.com/compute/docs/autoscaler/`.

[16] X. Li and C. Qian. An nfv orchestration framework for interference-free policy enforcement. In *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*, pages 649–658. IEEE, 2016.

[17] D. Liu, Y. Zhao, H. Xu, Y. Sun, D. Pei, J. Luo, X. Jing, and M. Feng. Opprentice: towards practical and automatic anomaly detection through machine learning. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, pages 211–224. ACM, 2015.

[18] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary. Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 98–106. IEEE, 2015.

[19] H. Mao, M. Alizadeh, I. Menache, and S. Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56. ACM, 2016.

[20] M. J. McGrath, V. Riccobene, G. Petralia, G. Xilouris, and M.-A. Kourtis. Performant deployment of a virtualised network functions in a data center environment using resource aware scheduling. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 1131–1132. IEEE, 2015.

[21] H. Moens and F. De Turck. Vnf-p: A model for efficient placement of virtualized network functions. In *Network and Service Management (CNSM), 2014 10th International Conference on*, pages 418–423. IEEE, 2014.

[22] K. Nagaraj, C. Killian, and J. Neville. Structured comparative analysis of systems logs to diagnose performance problems. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 26–26. USENIX Association, 2012.

[23] OpenStack. `http://www.openstack.org/`.

[24] OpenStack - HEAT. `https://www.openstack.org/software/releases/newton/components/heat`.

[25] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker. E2: a framework for nfv applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 121–136. ACM, 2015.

[26] RightScale. `http://www.rightscale.com/`.

[27] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 5. ACM, 2011.

[28] R. Shi et al. Mdp and machine learning-based cost-optimization of dynamic resource allocation for network function virtualization. In *Services Computing (SCC), 2015 IEEE International Conference on*, pages 65–73. IEEE, 2015.

[29] Squid. `http://www.squid-cache.org/`.

[30] Suricata. `https://suricata-ids.org/`.

[31] N. Vasić, D. Novaković, S. Miučin, D. Kostić, and R. Bianchini. Dejavu: accelerating resource allocation in virtualized environments. In *ACM SIGARCH computer architecture news*, volume 40, pages 423–436. ACM, 2012.

[32] X. Wang, C. Wu, F. Le, A. Liu, Z. Li, and F. Lau. Online vnf scaling in datacenters. *arXiv preprint arXiv:1604.01136*, 2016.

[33] Web Polygraph. `http://www.web-polygraph.org/`.

[34] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Sandpiper: Black-box and gray-box resource management for virtual machines. *Computer Networks*, 53(17):2923–2938, 2009.