Artificial Neural Network Model to Predict Affinity for Virtual Network Functions

Arthur Selle Jacobs, Ricardo José Pfitscher, Ricardo Luis dos Santos, Muriel Figueredo Franco,
Eder John Scheid, Lisandro Zambenedetti Granville
Institute of Informatics — Federal University of Rio Grande do Sul
{asjacobs, rjpfitscher, rlsantos, mffranco, ejscheid, granville}@inf.ufrgs.br

***Abstract*—Network Functions Virtualization (NFV) was proposed to migrate middleboxes that compose network services, such as firewalls and Network Address Translation (NAT) servers, from hardware to software running on Virtual Machines (VMs), commonly known as Virtualized Network Functions (VNFs). In NFV-enabled networks, VNFs can be chained in Forwarding Graphs (FGs) to provide services. These FGs establish the logical order in which network packets must traverse until reaching the end-service. In this scenario, network operators establish affinity and anti-affinity rules, which determine restrictions on the placement and chaining of VNFs according to how well or poorly VNFs operate together. To address the subject of identifying affinity relations in NFV-enabled networks, we previously proposed a mathematical model to measure the affinity between pairs of VNFs. However, that affinity model falls short for identifying affinity of VNFs not yet deployed, as they have no resource usage data to take into account. In this paper, we use artificial neural networks to predict affinity estimation for newly introduced VNFs, which still do not have usage data to be analyzed. This affinity neural network is trained using past affinity measurements, containing the data from VNFs, Physical Machines (PMs), and FGs of each measurement as features. We evaluate our solution by analyzing it over real usage data from a Cloud dataset, and conclude that neural networks can be used to provide affinity values for network operators, or NFV orchestrators, to plan the deployment of new VNFs.**

I. Introduction

Networks services are primarily composed of network functions, such as firewalls, Intrusion Detection Systems (IDSes), and Network Address Translation (NAT) servers, usually referred to as middleboxes [1]. These middleboxes are traditionally provided by proprietary hardware appliances, requiring constant infrastructure investments to keep up with technological advances. In addition, network operators have to acquire a multitude of specialized operational knowledge to manage such appliances. To address these issues, Network Functions Virtualization (NFV) was proposed. NFV migrates middleboxes from hardware to software running on Virtual Machines (VMs), commonly known as Virtualized Network Functions (VNFs). These VNFs can be deployed on Commercial-of-the-Shelf (COTS) servers, reducing operational and financial costs [2]. Further, by taking advantage of the virtualization technology, operators are able to rapidly make changes to services, reducing the ossification of the infrastructure, as well as increasing the scalability and flexibility of the network [3].

978-1-5386-3416-5/18/$31.00

In NFV-enabled networks, VNFs can be chained in Forwarding Graphs (FGs) to provide services. These FGs establish the logical in which order network packets must traverse the network until reaching the end-service. In this scenario, network operators establish affinity and anti-affinity rules, which determine restrictions on the placement or chaining of VNFs according to how well or poorly VNFs operate together. In this way, operators preserve the health of network services, as well as avoid the creation of bottlenecks, since misplacing a VNF can easily result on processing or forwarding bottlenecks [1]. Further, these affinity rules help network operators improve service performance and avoid resource contention [4]. Affinity rules can be based on a variety of criteria, such as VNFs resource usage, minimum hardware requirements, geolocation of data centers, or even specific needs of network operators [5]. Despite affinity being a critical subject in NFV-enabled networks, most efforts on this area are focused solely on establishing affinity relation based on resource usage of VNFs or VMs [6] [7] [8], disregarding aspects intrinsic to the health of NFV-enabled networks, such as latency and possible conflicts between VNFs.

To address the subject of identifying affinity relations in NFV-enabled networks, we proposed in a previous work [9] a mathematical model to measure the affinity between pairs of VNFs, based on an extensible set of criteria. This affinity measurement solution provides a way for network operators to easily identify issues in NFV networks, by finding anti-affinity relations. In the proposed affinity model, we divide the affinity criteria in two groups, regarding whether the analyzed VNFs are online or not: dynamic or static, respectively. When a new VNF is going to be inserted in an existing service, that VNF, by definition, is offline and will have no usage data for computing the dynamic criteria, and thus, the affinity will result solely from static information. Therefore, in this scenario, the affinity measurement might lack accuracy, since important information on determining affinity (the dynamic criteria) is not yet available.

In this paper, we propose an extension to the affinity measurement presented in [9], making use of an artificial neural network to predict an affinity measurement for newly introduced VNFs, which still do not have usage data to be analyzed. This affinity neural network is trained using historical affinity measurements, containing the data from VNFs, Physical Machines (PMs), and FGs of each measurement as

features. By taking advantage of neural networks, we estimate affinity values for the cases where the dynamic criteria is not yet available, virtually replacing the dynamic criteria in the affinity measurement model. To evaluate our solution, we use a large-scale real life Cloud environment trace from Google, which we convert into an NFV dataset, and make it publicly available for other researchers to take advantage of.

The remaining of this paper is organized as follows. In Section II, we present related work regarding affinity in NFV and network virtualization, as well as NFV solutions that take advantage of neural networks. In Section III, we introduce our solution for a history-aware affinity measurement, describing the affinity neural network and presenting a case study to illustrate a usage scenario of our model. In Section IV, we evaluate our solution by analyzing it over real usage data from an NFV dataset, converted from a Cloud trace. In Section V, we conclude this paper and present future work.

II. Related Work

Affinity and anti-affinity have been discussed to little extent in the NFV context. Most of current affinity solutions disregard properties intrinsic to NFV-enabled networks, such as bandwidth consumption and Service Level Agreements (SLAs). Meanwhile, artificial intelligence techniques, such as neural networks and machine learning, have been vastly explored in NFV, due to the advantages of efficiently identifying patterns and great generalization. Next, we discuss the relevant affinity-related solutions in NFV, as well as solution that take advantage of learning models.

Bouten *et al.* [10] propose a semantic model to define an ontology and rule set of affinity and anti-affinity constraints, which can be attached to Service Funtions Chains (SFC) by service providers for performance or economic reasons. The authors extend an existing virtualization description language to support such affinity constraints. The contributions of the work highlight the importance of affinity in NFV environments, and the benefits that may be sought from using them. However, in their work, affinity and anti-affinity relations are manually defined by service providers, requiring specialized knowledge of the network to identify such relations.

Riccobene *et al.* [11] propose an approach to address the challenges of deployment VNF with the required resources, based on identifying the quantity and types of resources to be allocated to a VNF during deployment time, and generating rules accordingly for an orchestrator to use. In addition, the authors propose an affinity characterization of VNFs by analyzing compute resource allocations and workload performance for each allocation, and a framework was developed to map those affinity relations. The work focuses on identifying affinity to aid on the creation of deployment rules, which is an excellent use-case for affinity relations. However, the authors focus solely on compute resource allocation to identify affinity, which can fall short for NFV environments, where network properties such as latency and bandwidth consumption affect directly the service quality. In this scenario, a history-aware affinity measurement, such as the one we proposed, could greatly enhance the creation of deployment rules.

Mijumbi *et al.* [12] propose a topology-aware, dynamic and autonomous system for resource allocation of VNFs based on a Graph Neural Network (GNN). A GNN is a supervised learning model aimed to solve problems in the graphical domain, which relies on two feed forward neural network to diffuse and exchange information of nodes in a graph. In this scenario, the nodes in the graph represent the historical requirements, dependencies, and utilization of VNF resources, whereas auxiliary neural networks take as input such resources observations to predict future requirements for each VNF in an SFC. This model makes intelligent use of neural networks to predict resource requirements using as input many features key to the affinity model we proposed. However, the authors are focused on guaranteeing that resource requirements are met for VNFs at all times, and thus do not consider any type of affinity relationship between VNFs to assure service quality.

Blenk *et. al.* [13] propose a novel procedure to improve online Virtual Network Embedding (VNE) runtime performance, relying on Recurrent Neural Networks (RNNs). Their contribution consists of an admission control mechanism, based on a RNN, to prevent VNE algorithms from spending time and resources trying to fulfill embedding requests that are not possible, or that cannot be completed in acceptable time. This work adequately illustrates the advantages of using machine learning techniques to enhance the performance of management tasks in virtual networks, as well as avoiding resource wastes and improving the network performance. However, as this solution is aimed at embedding virtual nodes and links on top of a network substrate, no considerations regarding affinity is made, despite permeating affinity-related subjects, such as resource allocation.

Even though affinity and affinity-related issues have been discussed by several authors, their approaches have focused mainly on computational resources awareness, and meeting resource requirements of VNFs. Consequently, these solutions fall short for avoiding anti-affinity relations to occur, since they do not take into account the dynamic behavior of online VNFs. Further, it is clear that the use of learning models, such as neural networks, can enhance NFV solutions, especially when identifying patterns, such as affinity relations. In the next section, we describe our history-aware affinity measurement that takes advantage of a neural network, trained with historical data to enhance affinity results.

III. Affinity Prediction Model

The concept of affinity refers to the correlation of different entities, representing their ability or inability to perform when combined in a certain way. Hence, we define affinity as an indicative of how well two VNFs operate, either when placed on the same PM or when chained on the same FG. In the following subsections we present a background on our work, providing a brief definition of our affinity model. In addition, we present the proposed affinity neural network model, which aims to predict affinity values before VNF deployment, and

a case study illustrating how a network operator can benefit from our prediction model.

A. Affinity Measurement Model

In our previous work [9], we proposed an extensible affinity model in which its possible to mathematically measure the affinity of a pair of VNFs based on a set of criteria. This set of criteria are divided into two types: static and dynamic. Static criteria refer to those that do not require the evaluated VNFs to be online. Information for static criteria are usually available in descriptors, such as the ones described by the European Telecommunications Standards Institute (ETSI) in [2]; and VNF templates, which consists of predefined VNFs configurations that are accessed to deploy new instances of a VNF [14]. Meanwhile, dynamic criteria relate to those that evaluate resource usage data (*e.g.*, CPU and memory usage) to measure affinity, and therefore can only be considered when VNFs are online.

Our proposed mathematical model relies on harmonic means to combine lower-level calculations into a single numerical value that represents the affinity between the two evaluated VNFs. In this model, each criterion has a scope, a type, and an affinity equation that results on a value (varying from 0.001 to 1) denoting how well the analyzed VNFs operate together regarding that criterion. In this way, the affinity measurement model is structured as a tree, in which the leaf equations are the affinity functions for each criterion, and the root node is the resulting affinity measurement. Using harmonic means to combine leaf-level equations keeps the final result value high in case the leaf-level results are high, and decreases the result value as leaf-level results decrease, avoiding low measures to be masked by any higher measure. Equation 1 presents the top-most equation in the affinity model, a harmonic mean between the static and dynamic affinity, represented by α_s and α_d respectively, where fg represents each FG that both VNFs are chained in. In addition, the resulting affinity, represented by $\alpha_{(vnf_a, vnf_b)}$, will have a different value for each FG that both VNFs are chained in, since the monitored data may differ for different chains.

$$\alpha_{(vnf_a, vnf_b)} = \frac{1+p}{\frac{1}{\alpha_s} + \frac{p}{\alpha_d}}, \quad \forall fg \in \{vnf_a \cap vnf_b\}$$

Where,

$$p = \begin{cases} 1 & \text{if the two VNFs are running,} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

By relying on harmonic means, the mathematical model is able to take into account outlier values that may be affecting the performance of the VNF's pair. Table I presents the set of criteria considered in the affinity model, as well as the scope and type of each criterion. More information regarding this set of criteria, as well as an in-depth description of the affinity measurement, is available on [9].

B. Affinity Neural Network

The focus of our previous work was to provide affinity measurements for running environments, and therefore detect misplaced VNFs and incompatible chains. However, such approach does not provide reliable results for the cases where VNFs are still not deployed in a service chain. This occurs because resulting affinity measurements of offline VNFs were based only on static information. Aiming to provide an estimation of the affinity for a given VNF, which has not yet been tested or deployed, and hence usage data (*e.g., memory or latency)* is also not available, we extend our previous model by introducing an artificial neural network to predict affinity values, based on historical affinity measurements. Such affinity estimations provide network operators, or NFV orchestrators, with more reliable information so to plan the migration and deployment of new VNFs and services.

Type	Scope	Criterion	Affinity Function
Static	PM	Minimum CPU	$0.001 \leq \alpha \leq 1.0$
		Minimum memory	$0.001 \leq \alpha \leq 1.0$
		Minimum storage	$0.001 \leq \alpha \leq 1.0$
	FG	NFV conflicts	$0.001 \leq \alpha \leq 1.0$
Dynamic	PM	CPU usage	$0.001 \leq \alpha \leq 1.0$
		Memory usage	$0.001 \leq \alpha \leq 1.0$
		Storage usage	$0.001 \leq \alpha \leq 1.0$
	FG	Bandwidth usage	$0.001 \leq \alpha \leq 1.0$
		Packet loss	$0.001 \leq \alpha \leq 1.0$
		Latency	$0.001 \leq \alpha \leq 1.0$

Table I: Set of criteria.

Artificial neural networks consist of learning models based on neuron-like units that interact through one-sided weighted connections. Through this process, neural networks are able to receive a generic number of inputs, referred to as features, and classify and predict a generic number of outputs, referred to as targets. Neural networks have at least two layers of neurons, the input and the output layers, which contain a variable amount of neurons to compute values. In addition, it is possible for neural networks to have hidden layers, with an arbitrary number of neurons to better classify and memorize possible results.

Neural networks have a high generalization rate, and hence can be applied to a multitude of different scenarios having well defined inputs and outputs. In addition, the simplicity and performance of the model, coupled with the ability to predict the values by learning from historical data make it a suitable candidate to address the problem we approach in this paper. The neural network designed to predict the affinity consists of four layers: one input layer, two hidden layers, and one output layer. Determining the optimal number of hidden layers, and the number of neurons in those layers are well-known problems in the neural network field, and do not have a straightforward solution. However, some authors [15] [16] claim that two hidden layers solve most of the problems, using

the same number of neurons as features for the first hidden layer, and half that amount for the second hidden layer. Hence, we follow that approach in our solution, and confirm it through a trial and error approach. The activation function used to compute values in the hidden layer neurons was the hyperbolic tangent, as shown in Figure 1, which was determined through empirical test data. In addition, the weights of each neuron is obtained through a gradient descent algorithm, relying on back propagation for computing the gradients [17], as most neural networks do.

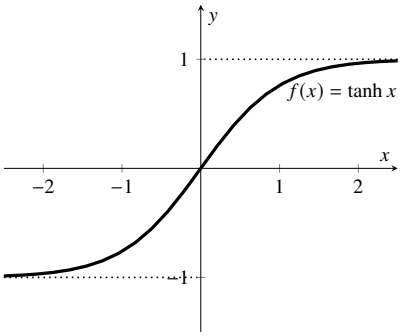

Figure 1: Hyperbolic tangent activation function.

As we aim to predict the affinity measurements using only information available in the static criteria, the features of our neural network use only values available from VNFs templates and descriptors. To maintain the number of features small, and to make it easier for the neural network to generalize the affinity relationships, we also provide as features values obtained from the affinity functions of static criteria. Below, we describe each feature used in the model. In total, ten features were used as input for the neural network, in which features 1 and 2 appear twice: once for each VNF of the pair being evaluated.

1) **VNF Type**: this value refers to the function provided by the VNFs evaluated. We mapped types of VNFs (*e.g.*, firewall, load balancer, cache) and associated them with a numerical value so they could be used as features in the neural network. The values of this feature vary between 0.1 and 0.9, where each decimal represents a different VNF type.
2) **VNF Scheduling Class**: numeric value that indicates how latency-sensitive a VNF is. This value ranges from 0 to 3, in which 3 is the most latency-sensitive (*e.g.*, VNF serving revenue-generating requests), and 0 is the least latency-sensitive (*e.g.*, non-business-critical analyses).
3) **Min. CPU Affinity**: result of Minimum CPU affinity function, from the criteria set shown in Table I. This affinity function takes into account the CPU specs available for both VNFs and compares them to requirements specified in VNF descriptors, resulting on a value between 0.001 and 1. For example, if two VNFs have a minimum CPU requirement of 750 MHz and the VMs in which the VNFs are hosted has only 500 MHz, this feature will result in a value of 0.6667.
4) **Min. Memory Affinity**: result of Minimum Memory affinity function, from the criteria set shown in Table I. This affinity function considers the amount of available memory for both VNFs and compares them to requirements specified in the VNFs descriptors, resulting on a value between 0.001 and 1. For instance, if two VNFs have a minimum memory requirement of 2048 MB and the VMs hosting the VNFs have 1024 MB of available memory, the resulting affinity for this feature will be 0.50.
5) **Min. Storage Affinity**: result of Minimum Storage affinity function, from the criteria set shown in Table I. This affinity function takes into account the available storage for both VNFs, and compares them to requirements specified in the VNFs descriptors, resulting on a value between 0.001 and 1. For example, if two VNFs have a minimum storage requirement of 500 I/O operation per second (IOPS) and the VMs hosting the VNFs can only perform 200 IOPS, the resulting affinity for this feature will be 0.4.
6) **Conflicts Affinity**: result of the NFV Conflicts affinity functions, presented in the criteria set from Table I. This affinity function evaluates both VNFs types and compares them to a predefined list of conflicts. The conflicts list contains information on what VNF types are known to generate bottlenecks if chained in a specific order. If both VNFs respect that conflicts list, the affinity result is 1. Otherwise, the affinity result is 0.001. For instance, if there is a predefined conflict that states that a Deep Packet Inspection (DPI) VNF should not be chained before a firewall VNF, and the two VNFs being evaluated are in fact a DPI that is chained before a firewall in the same FG, then this feature will result in value 0.0001.
7) **Same PM**: binary value representing whether or not the evaluated pair of VNFs are hosted in the same PM.
8) **FG Scheduling Class**: numeric value that indicates how latency-sensitive is the FG that both VNFs are chained in. This value ranges from 0 to 3, following the same logic as feature 2. However, if the VNFs are not chained in the same FG, this value receives -1.

This set of features is meant to represent a snapshot of all environment data related to those VNFs, both regarding their allocated resources and requirements, and their FGs, to provide more information for the neural network to find a better affinity match. In addition, all values for the features are normalized in a range between -1 and 1 before being introduced in the neural networks. Finally, the output layer in the neural network has only one neuron, which corresponds to the affinity value.

Figure 2 depicts a generic scenario in which our affinity neural network can be used, as well as how it fits in an existing NFV environment. Historical data from an NFV network is monitored and fed into our affinity measurement model, which stores all calculated affinities into a training database. Our

proposed solution evaluates the affinity of these historical data and feeds it to the affinity neural network to train it. Once this process is complete, the neural network is used to predict affinity of newly introduced VNFs. An algorithm then tests multiple scenarios, using the predicted affinities to determine the best placement and/or chaining for the new VNF, without the need of monitored usage data. In the following subsection, we present a case study to illustrate the feasibility of our solution.

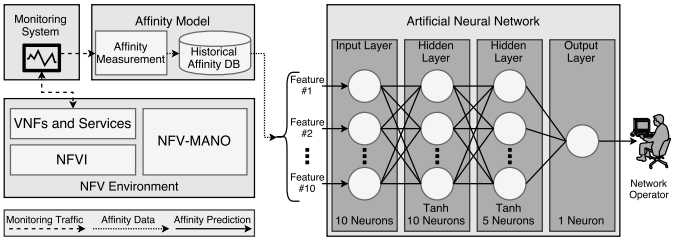

Figure 2: NFV Architecture with Affinity Neural Network.

C. Case Study

To illustrate the usage of our proposed solution, we present in this section a case study demonstrating how network operators may take advantage of an affinity measurement predicted by an artificial neural network. The scenario presented in Figure 3, includes an FG consisting of five VNFs: a firewall, a load balancer, two Intrusion Prevention Systems (IPSes), and a NAT server. In addition, the five VNFs are distributed among three different hosts. Tables II and III present the usage data for each PM hosting the VNFs, and the network usage information for the FG.

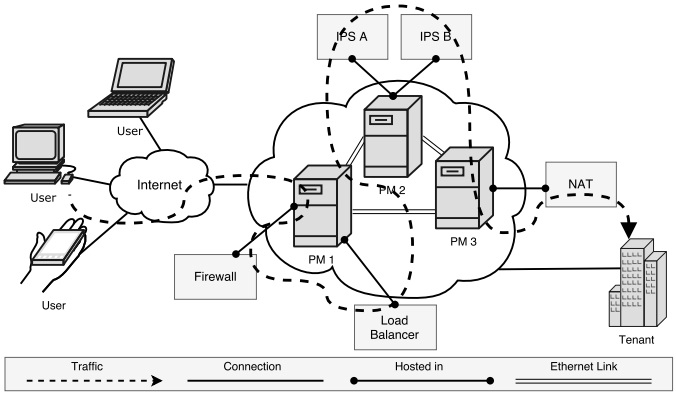

Figure 3: Case study NFV scenario.

Let us consider that a tenant needs to increase the security of the whole service. Thus, a network operator needs to introduce a new DPI VNF in this FG. In this scenario, the network operator does not have any usage information of the DPI, as it has not been deployed yet. Hence, it is difficult to infer the impact the new VNF may have on the service relying solely on the static information of the service, such as the service descriptor or the VNF template. Further, the position that the DPI is chained in the FG may affect the health of the service as well, resulting in processing or forwarding bottlenecks.

In the described scenario, an affinity metric, as the one we proposed in [9], could have been used to aid the collocation and chaining of this DPI. However, as the VNF has no usage data available yet, the affinity metric would not be able to prevent the creation of bottlenecks in the chain. For instance, we can see from Table II that the PM 2, hosting both IPSes, has a CPU usage of 60%. Hence, if the DPI is placed in the same host as the IPSes, resource contention on that host might increase, as the DPI and the IPS are both CPU-bound VNFs. Without having usage data from these VNFs, a network operator would have to guess the behavior of the VNFs, possibly leading to bottlenecks. Hence, by predicting affinity values through historical information, we can decrease the chances of such bottlenecks, without having to rely on a trial and error approach.

PM	CPU Usage	Memory Usage	Storage Usage	VNF	VNF CPU Usage	VNF Memory Usage	VNF Storage Usage
1	50%	40%	40%	Firewall	30%	20%	30%
				LB	20%	20%	10%
2	60%	50%	10%	IPS A	30%	25%	5%
				IPS B	30%	25%	5%
3	10%	10%	5%	NAT	10%	10%	5%

Table II: Case study PMs resources usage.

Flow	Traffic (Mbit/s)	Bandwidth Usage	Packet Loss	Latency (ms)
$Internet \rightarrow Firewall$	500	50%	1%	10
$Firewall \rightarrow LB$	200	25%	1%	1
$LB \rightarrow IPS_A$	100	25%	1%	5
$LB \rightarrow IPS_B$	100	37%	1%	5
$IPS_A \rightarrow NAT$	100	10%	1%	5
$IPS_B \rightarrow NAT$	100	37%	1%	5
$NAT \rightarrow Tenant$	200	20%	1%	10

Table III: Case study FG resources usage.

IV. Evaluation

In this section, we present a description of the prototype developed to evaluate our model, the dataset used to train, validate, and test the neural network, as well as the results achieved. In addition, we provide an in-depth explanation of the process we adopted in converting a real-life large-scale Cloud dataset, the Google Cluster trace, into an NFV dataset.

A. Implementation

To evaluate the proposed model, we developed a prototype in Python that implements the described affinity neural network. To train, validate, and test the learning model, we relied on measurements from an NFV dataset, converted from

the Google Cluster trace. The artificial neural network was implemented using the `MLPRegressor` object, from the Scikit-learn Python library for machine learning [18]. We run our development prototype on a server with 64 AMD Opteron(TM) Processor 6276 CPUs at 2300 MHz, 64GB of RAM, running a 4.4.0-66-generic Linux kernel. The source code of our implementation, as well as the results achieved, are publicly available in GitHub[1].

Algorithm 1 outlines the general behavior of the developed solution. Our implementation starts by initializing the neural network as described in Section III-B. Then, we parse the scenarios from a dataset consisting of real usage data, described further in Subsection IV-B. Having parsed all VNFs and FGs from the dataset, our algorithm produces every possible combination of VNF pairs that are either hosted on the same PM or chained in a FG, and calculates the affinity for each of those pairs. We then randomly split the affinity dataset into three mutually exclusive subsets [16]: 70% of VNFs pairs are used as the training database for our neural network; 15% incorporate a validation database, used to verify whether the training database is suitable; and the remaining 15% consists of the testing database, which are unknown test cases to evaluate our results. If the validation step of our algorithm does not produce predictions with an R-squared value higher than a manually defined threshold, the training dataset used would not have been representative enough. Hence, we start the dataset splitting process again, until the validation step produces a sufficiently accurate result. After this validation, our algorithm uses the trained neural network to test the cases that are still unknown to the training database.

To facilitate the reproduction of our results, we also published in GitHub a Python object of the affinity neural networks resulting from Algorithm 1. We also provide, in the source code, methods to dump and load the trained neural network object.

Algorithm 1 Affinity Neural Network Test

```
1:  procedure AffinityNeuralNetworkTest
2:      nn ← InitNeuralNetwork()
3:      dataset ← InitDataset()
4:      do
5:          fit_db ← Sample(dataset, 0.7)
6:          dataset_remain ← dataset − fit_db
7:          validate_db ← Sample(dataset_remain, 0.5)
8:          test_db ← dataset_remain − validate_db
9:          nn.Fit(fit_db)
10:     while nn.Validate(validate_db) <= threshold
11:     nn.Test(test_db)
12: end procedure
```

B. Cloud to NFV Dataset Conversion

As there is no large-scale dataset that contains VNFs monitored data that we could take advantage of in the literature, we used the trace data from the Google Cluster [19] to evaluate our model, which represents 29 days worth of monitored data. This dataset was chosen not only because of its rich collection of information on Cloud environments, but also because of the granularity of the data, since it is very similar to the data needed for the affinity measurement model. Unfortunately, as this is a Cloud dataset, some information regarding FGs and network usage data will have to be generated randomly to fit the affinity model.

The Google Cluster trace data represents mainly three entities: Machines, Jobs, and Tasks. A Machine constitutes a host for VMs to be executed. A Job describes a sequence of Tasks to be executed. A Task is the representation of a VM that must execute a goal. For each entity, the trace contains a series of life-cycle events, such as creation, update, and deletion, which contains resource usage and requirements in each event. In addition, the trace includes Machine attributes, describing information such as kernel version and clock speed, and Task constraints, which outline restrictions for each Task regarding the Machine attributes required for a Task to run. Since Machine attributes and Task constraints do not present any relevant information for the affinity model as it is, theses entries are disregarded.

Figure 4 depicts a visual representation of the conversion process we applied. Each life-cycle event in the trace has specific information about the Job, Task, or Machine associated. Due to complexity reasons, and aiming to achieve a more representative dataset, we choose to consider only insertion events of each entity, *i.e.*, the time that each entity started to execute, since they have the most information. To adapt the entities presented in the Google Cluster trace to an NFV environment, we consider a Job as an FG, since it establishes a logical order for task VMs to be executed, much like a sequence of VNFs in an FG. In addition, we consider Tasks as VNFs, since the dataset provides usage data for Tasks in a much similar granularity to the VNF data needed in the affinity model. Each VNF is given a random type, such as firewall or NAT server, loosely based on the resource consumption they present in the Google trace. VNFs of an FG are chained according to the order the corresponding Tasks have in their Job, and a synthetic traffic measurement is generated, according to a Normal distribution, between all VNFs in the chain. Finally, Machine events are considered PMs in which each VNF is deployed, and hence, PM information is aggregated to each VNF entry in the dataset.

As the Google Cluster trace consists of a massive amount of monitored data, with over 50 files of usage information, we only used a small portion of it due to time and processing constraints, converting only one of the 50 files into an NFV dataset. Further, since there are multiple entries of usage data for each task, due to complexity, we considered only the first usage data for each corresponding task during conversion. Anomalies, such as missing measurement fields, as described by Google, were disregarded during conversions to avoid inconsistencies in the dataset.

The dataset converted from Cloud to NFV environment,

[1]https://github.com/asjacobs92/vnf-affinity-history

used to evaluate our solution, as well as the script used for conversion, are publicly available in the project's GitHub repository[2]. If any other researcher would need more data from the Cloud dataset, the conversion script can be easily modified to consider more than just the one usage data file we converted. Table IV presents the resulting dataset after conversion. The decrease from the number of Tasks to VNFs is due to the inconsistencies found in the trace, which were discarded by the converter. The decrease from the number of Jobs to FGs is because we only maintained the Jobs which were related to the tasks we converted to VNFs. The decrease in the number of Machines is in part due to maintaining only the hosts in which the VNFs we converted were allocated. Another reason for the decrease in Machines is the fact that there is only one Machine event file for the entire 50 files of Jobs and Task trace. Hence, there are a significant amount of Machines that were only used for later Tasks in the trace.

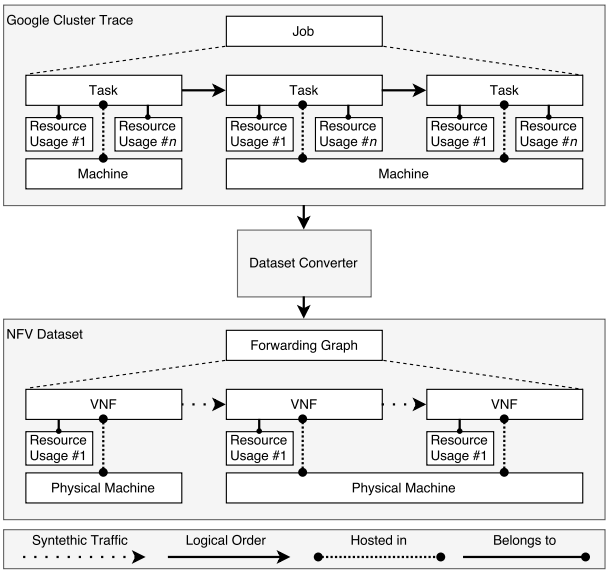

Figure 4: Google Cluster trace conversion to NFV dataset.

Google Cluster Trace File #1		NFV Dataset	
Entity	**# of Occurences**	**Entity**	**# of Occurences**
Jobs	4892	FGs	1694
Tasks	189994	VNFs	170510
Machines	21443	PMs	12478

Table IV: NFV dataset.

VNFs converted from the dataset were then combined in every pairing possible, finding all VNFs that were either hosted in the same PM or chained in the same FG, resulting on 1,302,524 pairs of VNFs. Having paired the VNFs, we took the affinity measurements considering both dynamic and static data, to train and evaluate the result of our neural network.

[2]The conversion process was executed using 64 different processes, in the previously described server, and it took five hours to complete.

Figure 5 presents a histogram of the calculated affinity for each pair of VNFs from the dataset that were either placed in the same PM or chained in the same FG. Is is important to point out that there is a large prevalence of higher affinities in the dataset.

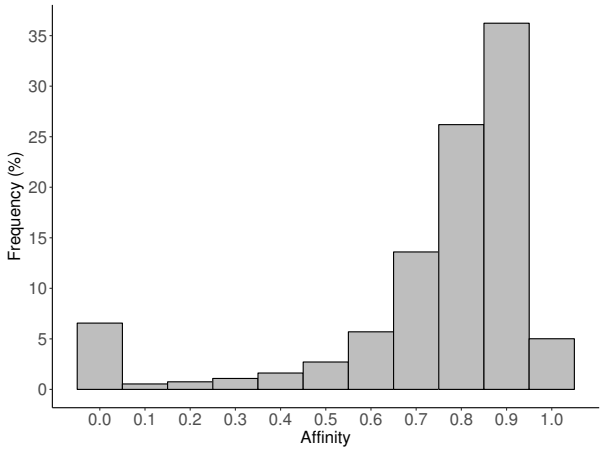

Figure 5: NFV Dataset affinity distribution.

C. Results

The results from our test cases can be seen in the graphs depicted in Figures 6 and 7. The first graph, presented in Figure 6, correlates the real affinity measurements with the static affinity measurements. Hence, the Y-axis consists of the affinity values calculated using only static information, whereas the X-axis represents the real affinity value for that same test case, calculated using static and dynamic information. In this way, the closer the grey dots are to the diagonal black line, the closer the static affinity is to the real affinity.

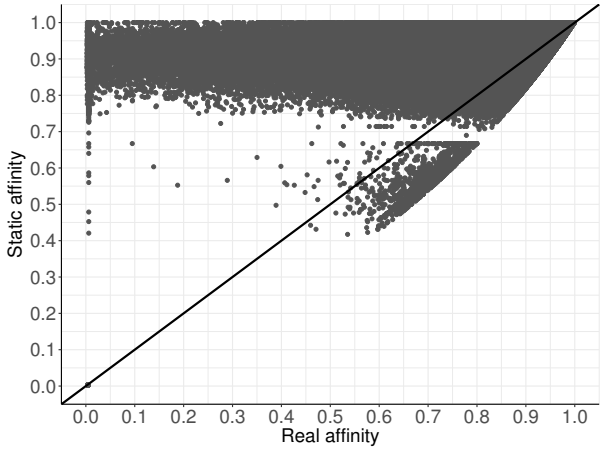

Figure 6: Static affinity and Real affinity correlation.

The graph presented in Figure 7 correlates the real affinity measurements with the predictions made by our affinity neural network. Therefore, the Y-axis represents the affinity prediction values the trained neural network produced, whereas the

X-axis represents the real affinity value for that test case. In this way, the closer the grey dots are to the diagonal black line, the closer the affinity prediction is to the real affinity. We can see that the affinity model using the neural network to predict affinity enhances the accuracy of our affinity measurements, in comparison to evaluating solely static criteria.

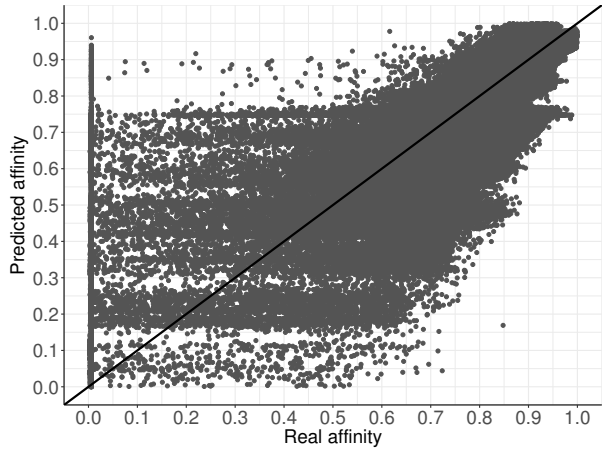

Figure 7: Predicted affinity and Real affinity correlation.

It is clear that, in general, the affinity measured using only static criteria resulted in high values, even in cases where the real affinity value was low. For the results presented in Figure 6, we calculated an R-squared value of 0.01867. This is due to the fact that the static criteria mainly evaluate if usage resources were met for each VNF. Since the Google Cluster takes into account resource requirements of Tasks when allocating them in Machines, the static affinity measurements resulted in high values. However, using the dynamic criteria, the anti-affinity relation of some VNF pairs are brought to light.

Using our affinity neural network, we are able to determine more accurately the affinity of VNFs before deployment. For the results presented in Figure 7, we calculated an R-squared value of 0.67, a much more accurate result than the one obtained with static affinity only. Analyzing the predicted affinity graph, it is clear that, for high affinity values, our model converges and very accurately predicts affinity for the VNFs involved. However, for lower affinity values the model does not perform so well. This is most likely due to the affinity distribution of our affinity dataset having a large number of high affinities. A more uniform affinity distribution might have resulted on more precise predictions.

Cases in which the neural network was not able to predict accurately affinity results consist of scenarios in which the static information available was not enough to represent the real affinity. For instance, if two cases in which the features of the neural network were similar, but the real affinity was divergent, the neural network would not be able to assimilate and generalize them. Therefore, having more representative static information, such as an indicator representing if a VNF is delay sensitive, throughput sensitive, loss sensitive, or jitter sensitive, could enhance predictions.

V. Conclusions and Future Work

In this paper, we extended our affinity measurement model proposed in [9], by introducing an artificial neural network to predict affinity in cases where resource usage data, that feed into the dynamic affinity criteria, are not available. In these cases, such as when a new VNF will be deployed into an existing service, a network operator, or an NFV orchestrator, can use this affinity neural network to predict the affinity of the new VNF regarding other chained VNFs, and with that information, determine the proper placement and/or chaining of the VNF. In addition, we presented a case study, demonstrating a scenario in which a network operator would benefit from our proposed solution.

We also converted a large-scale real world Cloud dataset, from the Google Cluster trace, to a comprehensive NFV dataset, and made it publicly available so that other researches may use it too. We then analyzed our proposed affinity neural network by training it with a fit database sampled from the converted NFV dataset, validating and testing it with cases not comprehended in the neural network training database. We analyzed the results of our evaluations and demonstrated that the proposed approach achieved an R-squared value of 0.67 when predicting affinity, a much more accurate result when compared to 0.01867 R-squared value the previous model achieved, which relied solely on static data when VNFs are offline. This evidence substantiates that neural networks are a suitable candidate for predicting affinity based on historical data.

As future work, we plan to introduce a chaining and deployment solution that relies on our affinity model to aid network operators on management and orchestration tasks. Also, we plan on developing a passive monitoring methodology based on existing logged information from VNFs, eliminating the need for an active monitoring solution, and reducing the deployment costs of our model. Finally, we plan to integrate other affinity criteria into the model to enhance the accuracy of our affinity measurements.

VI. Acknowledgement

We thank CNPq for the financial support. This research has been supported by call Universal 01/2016 (CNPq), project NFV Mentor process 423275/2016-0

References

[1] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys Tutorials*, 18(1):236–262, Firstquarter 2016.

[2] ETSI NFV ISG. Network Functions Virtualisation: Management and Orchestration. White Paper 1, December 2014.

[3] ETSI NFV ISG. Network Functions Virtualisation. White Paper 1, October 2012.

[4] S. Oechsner and A. Ripke. Flexible support of VNF placement functions in OpenStack. In *2015 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 1–6, April 2015.

[5] Alcatel-Lucent. Network Functions Virtualization - Challanges and Solutions. White paper, June 2013.

[6] J. Chen, K. Chiew, D. Ye, L. Zhu, and W. Chen. AAGA: Affinity-Aware Grouping for Allocation of Virtual Machines. In *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, pages 235–242, March 2013.

[7] J. Sonnek, J. Greensky, R. Reutiman, and A. Chandra. Starling: Minimizing Communication Overhead in Virtualized Computing Platforms Using Decentralized Affinity-Aware Migration. In *2010 39th International Conference on Parallel Processing*, pages 228–237, Sept 2010.

[8] F. Z. Yousaf and T. Taleb. Fine-grained resource-aware virtual network function management for 5G carrier cloud. *IEEE Network*, 30(2):110–115, March 2016.

[9] A. S. Jacobs, R. L. dos Santos, M. F. Franco, E. J. Scheid, R. J. Pfitscher, and L. Z. Granville. Affinity measurement for NFV-enabled networks: A criteria-based approach. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 125–133, May 2017.

[10] N. Bouten, M. Claeys, R. Mijumbi, J. Famaey, S. Latré, and J. Serrat. Semantic validation of affinity constrained service function chain requests. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 202–210, June 2016.

[11] V. Riccobene, M. J. McGrath, M. A. Kourtis, G. Xilouris, and H. Koumaras. Automated generation of VNF deployment rules using infrastructure affinity characterization. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 226–233, June 2016.

[12] R. Mijumbi, S. Hasija, S. Davy, A. Davy, B. Jennings, and R. Boutaba. A connectionist approach to dynamic resource management for virtualised network functions. In *2016 12th International Conference on Network and Service Management (CNSM)*, pages 1–9, Oct 2016.

[13] A. Blenk, P. Kalmbach, P. van der Smagt, and W. Kellerer. Boost online virtual network embedding: Using neural networks for admission control. In *2016 12th International Conference on Network and Service Management (CNSM)*, pages 10–18, Oct 2016.

[14] R. L. dos Santos, O. M. C. Rendon, J. A. Wickboldt, and L. Z. Granville. App2net: A platform to transfer and configure applications on programmable virtual networks. In *2015 IEEE Symposium on Computers and Communication (ISCC)*, pages 315–322, July 2015.

[15] A. Elisseeff and H. Paugam-Moisy. Size of Multilayer Networks for Exact Learning: Analytic Approach. In *Proceedings of the 9th International Conference on Neural Information Processing Systems*, NIPS'96, pages 162–168, Cambridge, MA, USA, 1996. MIT Press.

[16] David Kriesel. *A Brief Introduction to Neural Networks*. 2007. Available at http://www.dkriesel.com.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: Foundations of Research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[19] C. Reiss, J. Wilkes, and J. L. Hellerstein. Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, CA, USA, November 2011. Revised 2014-11-17 for version 2.1. Available at https://github.com/google/cluster-data.