Using Self-Organizing Maps for the Behavioral Analysis of Virtualized Network Functions

Giacomo Lanciano[2,1], Antonio Ritacco[1], Fabio Brau[1], Tommaso Cucinotta[1],
Marco Vannucci[1], Antonino Artale[3], Joao Barata[4], and Enrica Sposato[3]

[1] Scuola Superiore Sant'Anna, Pisa, Italy
{firstname.lastname}@santannapisa.it
[2] Scuola Normale Superiore, Pisa, Italy
giacomo.lanciano@sns.it
[3] Vodafone, Milan, Italy
{antonino.artale,enrica.sposato2}@vodafone.com
[4] Vodafone, Lisbon, Portugal
joao.oliveira3@vodafone.com

Abstract. Detecting anomalous behaviors in a network function virtualization infrastructure is of the utmost importance for network operators. In this paper, we propose a technique, based on Self-Organizing Maps, to address such problem by leveraging on the massive amount of historical system data that is typically available in these infrastructures. Indeed, our method consists of a joint analysis of system-level metrics, provided by the virtualized infrastructure monitoring system and referring to resource consumption patterns of the physical hosts and the virtual machines (or containers) that run on top of them, and application-level metrics, provided by the individual virtualized network functions monitoring subsystems and related to the performance levels of the individual applications. The implementation of our approach has been validated on real data coming from a subset of the Vodafone infrastructure for network function virtualization, where it is currently employed to support the decisions of data center operators. Experimental results show that our technique is capable of identifying specific points in space (i.e., components of the infrastructure) and time of the recent evolution of the monitored infrastructure that are worth to be investigated by human operators in order to keep the system running under expected conditions.

Keywords: Self-Organizing Maps · Machine Learning · Network Function Virtualization

1 Introduction

The novel *Network Function Virtualization (NFV)* paradigm [25] has been nowadays adopted by all the major network service providers in response to the increasingly demanding requirements they have to meet, in particular, in terms of performance, flexibility and resiliency. Indeed, traditional approaches that rely on the deployment of network functions on top of proprietary *specialized* physical appliances – typically sized for the peak-hour and very costly to maintain – are no more sustainable in the complex, fast-paced scenarios that can be found in modern telecommunication systems. Thanks to the amazing advances in the *cloud computing* space, having on-demand access to a diverse set of virtualized resources (computing, storage, networking, etc.) – running on commodity hardware – has never been so easy and convenient. In the context of NFV, this kind of virtualization technologies is leveraged according to the *private cloud computing* model, where general-purpose computing, networking and storage resources owned by the operator can be dynamically and automatically managed and orchestrated, to fit the needs of time-varying workloads. This allows for cutting costs and energy consumption, as well as shortening development cycles and time-to-market [16]. For example, a virtualized network infrastructure can be easily adapted to adequately support new products of an organization or, if customers request new network functions, all it takes to handle such requests is to spin up new VMs that can be rapidly decommissioned when the functions are no longer needed. In this way, network functions can be completely decoupled from the underlying physical appliances they are deployed onto and can be effectively developed as distributed, elastic, resilient software applications. For NFV data centers, the choice of private cloud infrastructures – as opposed to the use of public cloud services – is also corroborated by latency-related concerns. Indeed, since such service-chains are highly delay-sensitive (e.g., LTE, 4G), it is unpractical to rely on public cloud infrastructures, that are usually shared among multiple tenants and non-necessarily deployed according to the network operator needs.

In order to guarantee scalability, robustness to failure, high availability, low latency, virtualized network functions (VNFs) are typically designed as large-scale distributed systems [27], often partitioned and replicated among many geographically dislocated data centers. The larger the scale, the more operations teams have to deal with complex interactions among the various components, such that diagnosis and troubleshooting of possible issues become incredibly difficult tasks [12]. Also, the capacity of such systems is designed according to some technical and economical considerations, in order to support the *standard load* conditions under which the VNFs perform well, ensuring a number of diverse kinds of Service Level Agreements (SLAs) between network operators and their customers. However, when extraordinary events or cascade failures occur, the network is typically overloaded and the allocated resources are not sufficient anymore to process all the incoming flows. Monitoring the status of the data center through an efficient distributed monitoring infrastructure that continuously gathers system-level metrics from all the different levels of the architecture (e.g., physical hosts metrics, virtual machines metrics, application-level key performance indicators, event logs) is a necessary step in order to build a pro-active system capable of detecting signals of system overload in advance. Such data usually drives the decisions of human operators, for instance, in terms of which actions must be taken to restore the expected conditions of the system after an outage has occurred, or how the available components should be reconfigured to prevent possible SLA violations in case of an unexpected increase in the workload.

One of the major problems of data center operators is *anomaly detection*, i.e., pinpointing unexpected and/or suspect behaviors of the system whenever it significantly deviates from the normal conditions. Indeed, recognizing characteristic patterns of resource consumption in early stages can be crucial to avoid resource exhaustion and to redirect critical traffic peaks so to minimize the risk of SLA violations (i.e., such that human experts can focus their efforts on the most critical activities), or at least to alert the staff to prepare the remediation/mitigation procedures in advance. Even though the amount of data usually produced by NFV infrastructures is huge, most of it is not explicitly labeled by specialized personnel, so that *unsupervised* machine learning (ML) algorithms (i.e., clustering or vector quantization techniques) are the easiest ones to use, especially for anomaly detection purposes. The objective of these algorithms is to group data with a similar trend in macro-categories and allow operators to keep tens or hundreds of virtual machines under control at the same time.

1.1 Contributions

In this paper, we propose to use *Self-organizing Maps (SOMs)* to perform a behavioral pattern analysis of VM metrics aiming at providing a comprehensive overview of the major behavioral patterns and detecting possible anomalies in a data center for NFV. The technique can be used to perform a joint analysis of *system-level metrics* available from the infrastructure monitoring system and *application-level metrics* available from the individual VNFs. It aims at supporting data center operations and specifically capacity and performance monitoring, by providing insightful information on the behavioral patterns, in terms of resource consumption and exhibited performance, of the analyzed VNFs. In our approach, the SOM-based behavioral analysis is leveraged to deliver a sophisticated alerting subsystem, whose output can be directly consumed by human operators or could be used as a trigger for automated remediation procedures.

This paper constitutes an extended version of our prior work [6], where we added related background concepts and technical details on our technique, describing for the first time the non-Euclidean distance we adopted and the automated alerting system that we built downstream of the SOM-based analysis, and discussing additional experimental results obtained with the proposed approach.

1.2 Paper Organization

This paper is organized as follows. After discussing the related literature in Section 2 and some fundamental background concepts in Section 3, we present our approach in Section 4, along with the data processing workflow we designed for the massive data set available in the Vodafone infrastructure. In Section 5, we discuss some obtained experimental results that validate the approach and highlight its practical relevance. Section 6 concludes the paper with our final remarks and ideas for future research in the area.

2 Related Work

In this section, we briefly review some of the most related works that are found in the research literature on using ML, and SOMs in particular, for classification and anomaly detection in cloud and NFV data centers.

Anomaly detection can be framed as the problem of pinpointing unexpected and/or suspect behaviors of a system whenever it significantly deviates from the normal conditions. Similar problems can be found in other fields and applications such as, for instance: intrusion detection in cyber-security, machinery fault [30] and product quality issues detection [1] in industrial contexts, or fraud detection in finance [22]. It is important to stress that anomaly detection is, in general, an inherently imbalanced problem due to the scarcity of anomalous observations with respect to the ones related to the normal conditions of a system. In order to tackle this kind of challenges, a huge amount of solutions has been proposed that, depending on the scenario and the nature of the data to be processed, pose their foundations on well-established techniques coming, for instance, from the research fields of information theory and statistics.

In the recent years, ML techniques have been gaining more and more traction in the context of anomaly detection applications because of their proven effectiveness in many of the aforementioned scenarios. This is mainly due to the versatility of this kind of methods and the increasing availability of data from which they can learn from, in a continuous manner [3]. Most of the approaches to anomaly detection address the associated challenges by feeding ML models with counters like CPU utilization, memory contention and network-related metrics [12,31,34]. Others include also system-level and/or application-level event logs in the analysis to increase the amount of features and facilitate the extraction of relevant patterns [8,35]. Embedding textual information has been in fact made easier by the advancements in Natural Language Processing (NLP) research [2]. Few existing works also consider the need of assisting human operators in conducting root-cause analysis to be a highly desirable feature of anomaly detection systems [14,28].

One of the major roadblocks that can be encountered when applying ML for solving a task is the scarcity, or the complete absence, of labelled data, a very common scenario in many practical applications. Such issues can be overcome by employing so-called *unsupervised* learning techniques that, as the definition suggests, are designed to operate without a ground truth (i.e., annotated data). It is worth noticing that this characteristic of such class of learning algorithms has the side effect of increasing the amount of data that can be used for training an ML model. The principal application of unsupervised techniques is *clustering* that consists in the formation of groups (the *clusters*) of data samples that are similar, where similarity is defined according to the employed distance function.

A SOM is a particular kind of neural network that leverages on the *competitive learning* approach for cluster formation [17]. In this context, when a new sample is presented to the SOM during the training, the Best Matching Unit (BMU) – the closest neuron to the data sample according to the employed distance function – is selected and BMU and its neighbors are rewarded by updating their weights so to make them more similar to the selected sample. The iteration of this process leads to the formation of the clusters that are represented by the associate SOM neurons. SOMs are designed for mapping high-dimensional data into a lower-dimensional (typically 2-dimensional) space. One of the main characteristics of the obtained clustering is that it preserves the *topology* and *distribution* of training data, at clusters-level. In practice, it means that more clusters will be located in the more dense regions of the original domain (distribution) and that similar data samples will be associated to the same cluster or to neighbor clusters (topology).

In the context of anomaly detection, such approaches usually operate by building, starting from training data, a set of clusters of samples that are representative of the expected – normal – conditions of a system. After training, such model can be exploited to compare new patterns to known behaviors according to a predefined distance metric, in order to infer whether the observations are anomalous or not. In these applications, the above mentioned properties of retention of the original data topology and distribution give the SOM the capability of creating a suitable number of clusters for the most representative situations: this distribution of clusters allows for a more reliable characterization of anomalous patterns due to the higher granularity reserved to more common situations.

Since early 90s, SOMs – as a *neural* approach to clustering – have achieved remarkable results at processing industrial data [15] in different fields. In [7], a SOM–based system for the visualization of complex process dynamics is proposed. In this application, topology conservation enables a smooth visualization of non-linear process transitions in a 2–dimensional map and favors the understanding of the influence of process parameters on process behavior. Similar approaches that exploit dimensional reduction and visualization on an easy–to–interpret 2D map are used also in [10] for process monitoring purposes and in [11] where the aspects of visualization of the evolution of process conditions are handled. In [4], SOMs are used for the grouping of electrical components based on a wide set of features that are efficiently mapped in a low dimensional space. In [33], it is shown how a SOM–based system can be used for detecting anomalies within a steel plant as far as the process faults and product defects are concerned. In this case, SOM clusterization is used to group similar process conditions or product features that are subsequently labelled according to experimental tests. The ability of SOMs to yield a distribution of the clusters in the problem domain that faithfully reflects the observed phenomenon behavior allows the generation of more specific clusters in the denser regions of the domain. The capability of SOMs of managing high–dimensional data and mapping them into a lower dimensional one have been exploited in medicine as well. In

[5], sonographic signals are processed and grouped in order to characterize those ones associated to breast cancer diagnosis. Another SOM-based approach was used in [32] to allow the analysis of complex cytometry data that is hard from a point of view of human experts due to the huge amount of variables to be taken simultaneously into consideration.

For what concerns NFV applications, the existing literature reports that ML techniques have been effectively used to solve different problems. In particular, in [13] a set of ML techniques are tested for an anomaly detection application. In this case, though, only supervised methods are considered and their performance is compared on data sets containing NFV features associated to different types of faults. Similarly, in [24], a *supervised* SOM-based method is proposed for fault detection. Here, a SOM is used to cluster *labelled* data, annotated by human experts to state which clusters correspond to faulty conditions, related to NFV performance indicators. In [26], SOM-based and other general clustering techniques are used for the same purpose in a small test-bed in the context of NFV. Likewise, in [20], the popular K–means algorithm is used to cluster cells traffic data in order group cells with similar through–time behavior and enable optimizations in the use of resources.

3 Background Concepts

3.1 Self-Organizing Maps

A SOM is an unsupervised vector quantization technique, used to produce a topology-preserving map using a competitive learning algorithm. The aim of the SOM training algorithm is to encode a data manifold (e.g., a sub-manifold $V \subseteq \mathbb{R}^N$) into a finite set $W = \{w_1, \cdots, w_M\}$ of reference vectors where $w_i \in \mathbb{R}^N$ is called *codebook*. Formally, a SOM is defined by a pair of maps (w, b). $w : \mathcal{L} \to \mathbb{R}^N$ is a discrete map from a two-dimensional lattice into a finite vector space, a.k.a., *features space*. Recall that a two-dimensional lattice of dimensions $H \times K$ is a discrete set

$$\mathcal{L} = \{hA + kB \,|\, h < H,\, k < K,\, h, k \in \mathbb{N}\} \subseteq \mathbb{R}^2$$

where $A, B \in \mathbb{R}^2$ determine its shape (e.g., $A = (1, 0)$ and $B = (0, 1)$ produce a *rectangular* grid, whereas $A = (\frac{1}{2}, \frac{\sqrt{3}}{2})$ and $B = (1, 0)$ produce an *hexagonal* grid). For the sake of simplicity, $\mathcal{L}$ is indexed with a lexicographical order (from 1 to $H \times K$), its elements $r_i \in \mathbb{R}^2$ are called *units* or also *neurons* and the images $w_i = w(r_i)$ of the neurons in the features space are called *weights*. Given a sample vector $x \in V$, $b : \mathbb{R}^N \to \mathcal{L}$ returns the *best-matching-unit* (BMU) i.e., the unit whose weight is closest to the input sample (or any such units, if multiple ones exist) depending on a distance d in the feature space.

$$b(x) \in \operatorname*{arg\,min}_{r \in \mathcal{L}} d(x, w(r)) \tag{1}$$

A common choice for the distance d is the Euclidean distance, albeit alternative choices are possible (e.g., see the discussion later in section 4.3).

For each training epoch t, an update to the SOM weights is performed, for each input sample, as follows. At each iteration t', an *input data* x is fetched (using either a random or a sequential scheduling) and its associated best matching neuron $b(x)$ is computed. Then, the weights of *all neurons* are updated according to (2), where h is called *neighborhood function* and is defined as (3) (assumed to be a Gaussian in what follows).

$$w_k^{(t'+1)} = w_k^{(t')} + \alpha(t)h(b(x), r_k, t)(x - w_k^{(t')}) \quad \forall k \tag{2}$$

$$h(r, s, t) = -\exp\left(\frac{\|r - s\|^2}{\delta(t)}\right) \quad \forall r, s \in \mathcal{L} \tag{3}$$

Here, $\alpha(t')$ and $\delta(t')$ are respectively the learning rate and the radius of the neighborhood function, which depend on the current epoch t ($\alpha, \delta : \mathbb{N} \to \mathbb{R}$), and decrease across epochs either linearly or exponentially, to make the algorithm converge. It is important to notice that, for each training sample x, not only the winning reference is modified, but the adaptation to x affects all the weights w_j depending on the proximity of r_j to $b(x)$ with a step size that decreases with the distance between the units r_j and $b(x)$ in the lattice. This way neighboring units respond to similar input patterns and each data point close in the input space is mapped to same or nearby map neurons (inducing a topology-preserving property on the codebook). The weights of the neurons w_i are typically initialized either by randomly sampling the V data set or using the well-known Principal Components Analysis (PCA) technique.

A key difference between the SOM training algorithm and other vector quantization or clustering techniques is that, in the neighborhood function definition (3), the topological distance between a pair of units is declined as the Euclidean distance on the map and not in the data space. The formulation in (2) is called the *online update* rule, that is not suitable for a parallel implementation since each iteration directly depends on the one immediately before and only processes a single data sample at a time. Therefore, a *batch parallel* implementation has been proposed: instead of updating the neuron weights for each data sample, they are updated after a batch $B \subseteq V$ of N' data samples (in the following we will assume $N' = N$, for the sake of simplicity). Essentially, the term of (2) that depends for each $r_k \in \mathcal{L}$ on the input sample by $h(b(x), r_k, t)(x - w_k)$ is replaced by a weighted sum of the same terms computed in parallel for all samples in the batch, using the formula:

$$\frac{\sum_{x \in B} h(b(x), r_k, t)\,(x - w_k)}{\sum_{x \in B} h(b(x), r_k, t)} \quad \forall k \tag{4}$$

This way, one can compute in parallel all numerator and denominator parts (i.e., $h(b(x), r_k, t)(x - w_k)$ and $h(b(x), r_k, t)$, respectively) for each sample in each batch and then sum up all numerator and denominator parts and finally compute the weight update.

After the training is complete, the result is that the manifold data V is divided into a finite number of subregions:

$$V_i = \left\{x \in V |\, \|x - w_i\|_2 \leq \|x - w_j\|_2 \quad \forall j \neq i\right\} \tag{5}$$

called Voronoi tessellation, in which each sample vector x is described by the corresponding weight of the BMU $w(b(x))$. It is important to point out that the update rule is fundamental to the formation of the topographically ordered map. In fact, the weights are not modified independently of each other but as topologically related subsets. For each step a subset of neurons is selected on the basis of the neighborhood of the current winning unit. Hence, topological information is supplied to the map because both the winning unit and its lattice neighbors receive similar weights updates that allow them, after learning, to respond to similar inputs. After the training phase, the map can be used to visualize different features of the codebook and of the represented data, such as (i) the density of the reference vectors (e.g., with a color scale proportional to neuron hits); (ii) likewise, the distances among reference vectors, where a dark color indicates a large distance between adjacent units, and a light color indicates a small distance (i.e., the so-called U-matrix); (iii) a plot of the reference vectors for each neuron, to see at a glance all the different behaviors detected in the training dataset.

A careful choice of the SOM hyper-parameters should be made in order to have a suitable trade-off in terms of quality of the clustering and computational performance. Some details on how the right hyper-parameters have been chosen for our analysis are given in Section 5. In order to mitigate the problem of having different neurons specializing on almost the same data samples (e.g., when the number of SOM neurons is large with respect to the data sample variability), we have applied an automated grouping technique over the SOM reference codebook, detailed in Section 5.4.

3.2 VMware vRealize Operations Manager

The vRealize Operations Manager (vROps)[5] – by VMware – is an enterprise-grade software used at Vodafone to operate the NFV infrastructure. Such framework can be deployed either on-premise or in the cloud and its main purpose is to support operations teams in automating and continuously improving their fundamental activities, also leveraging on data-driven methodologies. Indeed, the core of vROps consists of a pervasive monitoring infrastructure that collects system data at every level of the stack (e.g., physical hosts, virtual machines, networking components, etc.) and feed them to a powerful analytical engine that is able to provide useful insights and actionable feedback to the human operators, such that possible issues or anomalies can be early spotted and corrected. More than 300 system metrics, being them classical raw counters (e.g., cpu utilization, memory contention, network traffic, etc.) or more convoluted analytics computed by the engine, can be exported from the system, allowing also for the integration with third-party tools. Besides monitoring and alerting functionalities, vROps enables automated management of the VMs (or containers) that compose the deployed applications such that, for instance, the corresponding

[5] https://docs.vmware.com/en/vRealize-Operations/index.html

workloads can be balanced according to the optimization of specified indicators (e.g., application KPIs, licensing costs, etc.).

4 Proposed Approach

In this paper, we propose the use of SOMs in order to perform a behavioral analysis of the VMs that implement VNFs within an NFV data center infrastructure. Our approach consists of the joint analysis of two classes of metrics that are usually collected and analyzed independently of one another: *system-level metrics*, reporting information related to the utilization of the underlying infrastructure, hereafter also referred to as *INFRA* metrics, which are usually available through the NFV infrastructure manager (e.g., the well-known VMware vRealize Operations Manager or others); and *application-level metrics*, i.e., KPIs of the individual virtualized services, collected through their own monitoring subsystems, which will be referred to as *VNF* metrics. Considering both types of metrics allows for gathering a comprehensive overview of the major behavioral patterns that characterize VMs and possibly identifying suspect (anomalous) behaviors.

The proposed technique relies on the capability of SOMs to preserve the topology in the projection from the input space to the SOM reference vector space. In other words, using SOMs similar input patterns are captured by same or nearby neurons (see section 3.1 for details). A VM behavior can be monitored by considering the shift of its BMU, during the time horizon under analysis, so that any changes in a *suspect* BMU could be used to trigger an alarm.

4.1 Workflow

We realized a SOM-based clustering tool that is capable of detecting anomalies by clustering using a number of input metrics. In our experimentation, we have been applying this technique over individual monthly data available with a 5-minutes granularity (288 samples per day, per metric, per monitored VM or

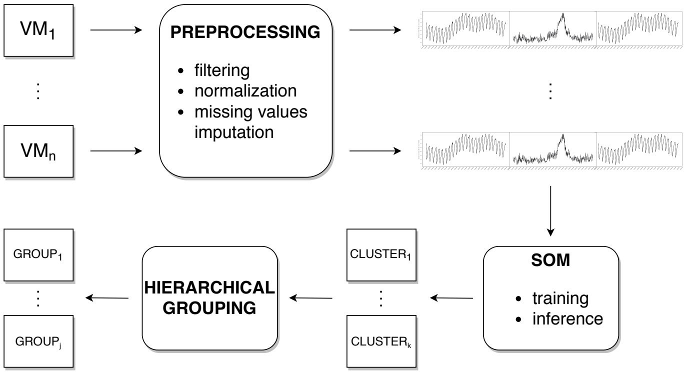


Fig. 1: Overview of the SOM-based clustering workflow.

physical host), amounting to several GBs of data per month, for a specific region of the Vodafone network operator. Figure 1 summarizes the overall workflow that we applied to process the available input metrics. First, the raw data are preprocessed to address possible data-quality issues (e.g., missing values imputation and time-series detrendization) and to filter out the additional information that is not relevant for the analysis. The input samples to the SOM are constructed, for each VM, by dividing the time horizon under analysis according to a predefined period (i.e., 24 hours) and merging the contributions of the individual metrics into a single vector. Then, such data are fed to the SOM that, after a training phase, infers for each VM the neuron capturing the most similar behavior and, thus, clusters on the various behavioral patterns of all the various VMs under analysis.

The input data are filtered – on the k specified metrics – and partitioned to have a sample (i.e., a time-series) for each metric, VM and period (usually a day) of the time horizon under analysis. Before being fed as input to the SOM training phase, samples are subject to a preprocessing phase, addressing possible issues such as (i) missing values and (ii) significant differences in the magnitude among the different metrics. On the one hand, to address (i), a data imputation strategy (i.e., a simple linear interpolation) is performed to mitigate the absence of data points within a sample and to retain as much data as possible for the analysis. However, in order to preserve the quality of the data set, the interpolation step has been designed not to be *aggressive*, such that a time-series can be discarded if it contains too much consecutive missing values. On the other hand, it is recommended to address (ii) when using SOM for multi-metric analysis since, due to the Euclidean distance being used as samples distance evaluation mechanism, metrics with significantly larger values (e.g., number of transmitted/received packets or bytes) tend to hide the contribution of other metrics which take on smaller values, for instance, being bounded by a predefined range that is much smaller (e.g., CPU utilization percentage). We have designed two possible strategies to tackle such problem. The first strategy, referred to as *normalized*, consists of computing the so-called *z-score*, i.e., scaling each time-series by subtracting its mean and dividing by its standard deviation. Using such a strategy hides any information regarding the magnitude of the original values and emphasizes differences in shapes. The second strategy, referred to as *non-normalized*, consists of scaling each time-series to the $[0, 1]$ range of values considering, for each metric, the historically observed minima and maxima values. Such a strategy retains information regarding the magnitude of the original values while keeping the data bounded in the same interval. However, this technique causes different metric patterns with very similar shape, but differing merely in their magnitude, to be grouped into different SOM neurons at a certain distance from each other (in the SOM grid topology). Depending on the chosen strategy, we obtain either an analysis focused on the shapes of the behavioral patterns, or we can also distinguish among the absolute values of the average levels of the metrics. In general, in the latter case one should expect more clusters to be outputted with respect to the former case, due to the possibility that the system could have experienced very diverse levels of load during its operation. Hence, one should take this possibility into account and increase the size of the SOM grid when performing a non-normalized analysis in order to avoid neurons *over-population* (i.e., too many patterns crowding within the same BMU), despite them being significantly distant from each other.

Each input sample to the SOM is constructed by concatenating k vectors (one for each of the k metrics under analysis), for each VM and period. Notice that, since INFRA metrics have been provided with a 5-minutes collection granularity, if a period of a day is considered, we typically have for each day 288 data points of each metric and for each VM. After the training phase, the SOM is used to infer the BMU for each input sample, i.e., the neuron that exhibits the least quantization error when compared with the considered input sample. Multiple VMs are expected to be associated to the same BMU and, thus, a number of different VM clusters can be derived from such process. Such an output can be used by a data center operator to visually inspect the behaviors assumed by the different VMs during the time horizon under analysis, in order to spot possible suspect/anomalous ones. Furthermore, since the individual input samples are related to the behavior of a specific VM at specific point in time, it is also possible to visualize the evolution of the VMs throughout the time horizon, to possibly detect interesting patterns in their behavioral changes.

On top of the clustering mechanism described above, we have devised an approach capable of detecting possible suspect behaviors without the need for a human operator to daily inspect the status of the SOM (these aspects are described in details in Section 4.4). Such additional feature consists of an alerting system that is triggered whenever an input sample is firstly associated to a group of similar neurons but in the following days a sudden group change takes place. Because of the considerable distance from the BMU (i.e., the *closest* neuron) of the neurons in the two different groups, such samples are likely to depict an uncommon behavior and, thus, an alert is raised to the operator. Besides the aforementioned support that such a tool can give to data center operators in their manual operations, this feature in particular enables the possibility to deploy a fully automated anomaly detection system.

4.2 SOM Implementation

To implement our anomaly detection tool we leveraged on an efficient open-source SOM implementation, namely `Somoclu`[6], which has been designed around the batch parallel SOM variant (see section 3.1) to employ multi-core acceleration, as well as GPGPU hardware acceleration, to perform massively parallel computations [36]. Such accelerations have been proved to be necessary in order to reach a satisfactory performance when tackling the massive data set provided by Vodafone. In the future, we plan to switch to a new implementation we recently realized performing even better [23].

[6] https://github.com/peterwittek/somoclu

4.3 Hierarchical Grouping

An interesting aspect that came to our attention during the development of the aforementioned SOM-based approach is that, whenever using relatively big SOM networks, the training phase ends up with many close-by SOM neurons catching behaviors that were very similar to each other. This is in line with the topology-preservation property of the SOMs, i.e., close-by input vectors in the input space are mapped to close-by neurons in the SOM grid. This phenomenon can be controlled to some extent by acting on the neighborhood radius. However, from the viewpoint of data center operators, a set of close-by neurons with relatively similar weight vectors needs to be considered as a single behavioral cluster/group. For this reason, after the SOM processing stage, we added a step consisting of a top-down clustering strategy, based on recursively separating weight-vector's sets whose diameter is higher than a given threshold. The principal aim of this technique is to offer the possibility of collapsing similar SOM neurons, according to the distances among their representative vectors, in order to decrease the possibility to raise an alarm when it is not needed (e.g., consider very frequent movements of a VM between two similar neurons over time) and to facilitate the human operators in interpreting the results and spotting anomalous behaviors. Indeed, as shown in Section 5.4, this led to the overall technique outputting a reduced and more comprehensible number of behavioral clusters.

Specifically, the aforementioned technique, known as *hierarchical clustering*, can be described as follows. Let ε be a fixed threshold which provides a bound for the maximum diameter of a group. The algorithm consists of the following steps:

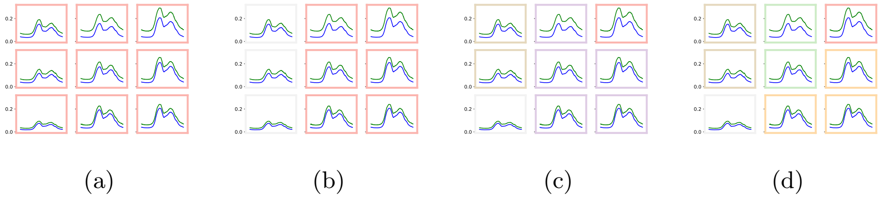
1. **Initialization:** The set of the groups to process is initialized with a single group $\mathcal{G}_0$ containing all the neurons $\mathcal{G}_0 = \{n_1, \cdots, n_{H\times K}\}$, and the set of the final groups is initialized to be an empty set.
2. **Distance Measure:** A group $\mathcal{G}$ is removed from the set of groups to process and its diameter D is computed by finding the two farthest away neurons:

$$(n_S,\, n_N) \in \underset{(n,m)\in\mathcal{G}}{\arg\max}\, d(w(n), w(m)), \qquad D = d(w(n_S), w(n_N))$$

   where $w(n)$ is the weight of neuron n. In the case that $\mathcal{G}$ contains just one neuron, its diameter D is defined as zero.
3. **Splitting:** If $D \leq \varepsilon$ (i.e., the diameter is within the specified threshold), then $\mathcal{G}$ is moved to the set of final groups. Otherwise, the group is split into two smaller (non-empty by construction) groups $\mathcal{G}_1$ and $\mathcal{G}_2$ defined as:

$$\mathcal{G}_1 := \{n \in \mathcal{G} \,:\, d(w(n), w(n_S)) \leq d(w(n), w(n_N))\}$$
$$\mathcal{G}_2 := \{n \in \mathcal{G} \,:\, d(w(n), w(n_S)) > d(w(n), w(n_N))\}$$

   that are added to the set of groups to process.
4. **Loop:** Steps 2,3 are repeatedly applied to all the elements in the set of groups to process, until it becomes empty, and the set of final groups contains only groups with a diameter lower than or equal to ε.



(a) (b) (c) (d)

Fig. 2: Example of grouping steps: (a) initialization; (b) first split; (c) second split; (d) final split. Neurons with same border color belong to the same group.

Figure 2 reports a graphical representation of how the algorithm works on a real example. From left to right, we can see all the four steps of the algorithm that bring to the final result in which each group contains only neurons with a pair-wise distance smaller than the provided threshold.

As explained above, the kernel of the hierarchical clustering technique is the measure of the diameter of a set. This implies that the definition of the distance impacts on the final result.

Definition 1. *For each $p \in \mathbb{N}$, the function $d_p : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_+$ defined as*

$$d_p(v, w) := \left(\sum_i |v_i - w_i|^p \right)^{\frac{1}{p}}$$

is the Minkowski distance of order p.

Note that, according to Definition 1, the Minkowski distance with $p = 2$ is the Euclidean distance, and that d_∞ degenerates into the Chebychev distance (maximum among the coordinates). Figure 3 shows that using $p = 4$, or in general a value higher than 2, allows for increasing the distance between neurons exhibiting a spike (i.e., neurons that are almost flat, except for an isolated huge value), so that we are able to isolate in a dedicated group such spiky neurons.

4.4 Alerting

A grouped SOM grid combined with a *calendar* representation of the VM behaviors can be used by an operator to spot possible anomalies. A calendar representation is a table containing for each couple (VM, DAY) a reference to the corresponding group. In addition, we designed a set of alerting systems based on heuristic methods, that can be used to simplify the inspection of such behaviors. We propose two main categories of alerting systems: the *calendar-view* alerting system, consisting of techniques that give a global view of the alerts over the entire period of interest, and the *dashboard-like* alerting system, consisting of techniques that give a detailed view of the behaviors that raise the alerts. In what follows, $V = \{v_1, v_2, \ldots, v_i, \ldots\}$ is the set of virtual machines under analysis, $D = \{d_1, d_2, \ldots, d_i, \ldots\}$ is the set of days that compose the time period

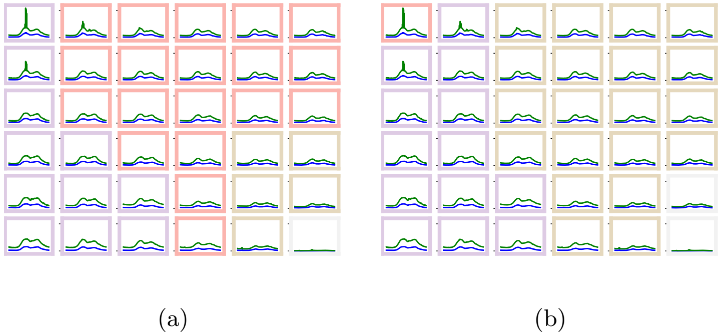

(a) (b)

Fig. 3: Examples of grouping using different p values. Grouping with $p = 2$ (a) makes no distinction between spiky and smooth neurons, whereas grouping with $p = 4$ (b) clusters the spiky neuron on the top-left corner in a dedicated group.

under analysis and grp : $V \times D \to G$ is the function that associates to each couple (VM, DAY) the corresponding group.

Calendar-View Alerting System This category contains those alerts that generate a calendar table in which each couple (VM, DAY) is associated with a value between 0 and 1, providing a level of alerting. In what follows, we denote with p the *period* and with m the *memory*, both expressed in days (i.e., $p = 7$ days, $m = 2$ weeks).

Definition 2 (Strong). *Given p, m, the Alert takes one VM v and one day d and returns a boolean value raising an alert if the VM v is classified into a different group in at least one day among the ones at most m periods apart:*

$$\mathit{Alert}_s(v,d)\ :\ \text{”}\exists j \in \{\pm 1, \cdots, \pm m\}, \quad \mathit{grp}(v,d) \neq \mathit{grp}(v,d-jp)\text{”} \tag{SAS}$$

This alerting system is the most peaky (i.e., often producing false-positives) and, thus, performs the best when used in contexts where a few changes occur.

Definition 3 (Weak). *Given p, m, the Alert takes one VM v and one day d and returns a boolean value raising an alert if the VM v is classified into a different neuron in all the days among the ones at most m periods apart:*

$$\mathit{Alert}_w(v,d)\ :\ \text{“}\forall j \in \{\pm 1, \cdots, \pm m\}, \quad \mathit{grp}(v,d) \neq \mathit{grp}(v,d-jp)\text{“} \tag{WAS}$$

This alerting system is more loose than the previous – sometimes producing false-negatives – and, thus, performs the best in chaotic contexts, where many random changes occur.

Definition 4 (Fuzzy). *Given p, the Alert takes one VM v and one day d and returns a real number, between 0 and 1, defined as follows:*

$$\mathit{Alert}_z(v,d) := \quad \#\{j \in \mathbb{Z}\ :\ \mathit{grp}(v,d) \neq \mathit{grp}(v,d-jp)\}/\#D(v,d) \tag{ZAS}$$

where $D(v,d) = \{j \in \mathbb{Z} \,:\, \exists grp(v, d - jp)\}$ *is the set of all the comparable days.*

This alerting system, producing real values, can be used in a wide range of situations and could be useful to understand if a change in the behavior of a VM is common or infrequent.

Dashboard-Like Alerting System The aim of a dashboard-like alerting system is to provide a detailed view of the behaviors which raise the alert, providing also further information on the geometrical distance between the actual and the expected behavior in terms of weight of the SOM or also a count of the frequency of VM/Days which are clustered into *rare* groups.

Definition 5 (Expected Behavior). *Let* v *be a VM for which an alert is raised at day* d*, i.e.,* $Alert(v,d) = 1$*. Let* $\tilde{d}$ *be the nearest day, corresponding to the same weekday, for which the most common group is taken from* v *and for which* $grp(v,d) \neq grp(\tilde{d})$ *holds. Then, we define*

- $\mathtt{GRP}{:=}\ grp(v,d)$
- $\mathtt{NEU}{:=}\ neu(v,d)$
- $\mathtt{E\_GRP}{:=}\ grp(v,\tilde{d})$
- $\mathtt{E\_NEU}{:=}\ neu(v,\tilde{d})$
- $\mathtt{DIST}{:=}\ \|w(\mathtt{E\_NEU}) - w(\mathtt{NEU})\|_2$

where the function $neu : V \times D \to \mathcal{L}$ *returns the coordinates of the BMU associated to the behavior of a VM* v *during a day* d *and the function* w*, defined in section 3.1, returns the weight of a neuron.*

Such alerting system depends on the output of the calendar-like alerting system. Usually, we apply this method to the *weak* alerting system table (see Definition 3) in order to avoid false-positives alerts.

Definition 6 (Suspicious-Day). *Given a parameter* K*, let* $occ_d : G \to \mathbb{N}$ *be the function that counts the occurrences of a group in the days.*

$$occ_d(g) := \#\{d \in D \,:\, \exists v \in V,\, grp(v,d) = g\}. \tag{6}$$

If a group g *is such that* $occ_d(g) \leq K$*, then those VMs whose take the group* g *are stored in a table whose columns are* `DAY, VM, NEU, GRP, OCC_DAY`*, where* `OCC_DAY`$= occ_d(g)$

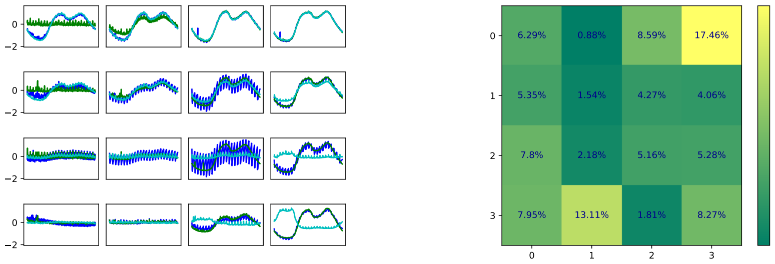
Such alerting system helps in catching days in which an infrequent group appears.

Definition 7 (Suspicious-VM). *Given a parameter* K*, let* $occ_v : G \to \mathbb{N}$ *be the function that counts the occurrences of a group in the VMs.*

$$occ_v(g) := \#\{v \in V \,:\, \exists d \in D,\, grp(v,d) = g\}. \tag{7}$$

If a group g *is such that* $occ_v(g) \leq K$*, then those VMs whose take the group* g *are stored in a table whose columns are* `VM, DAY, NEU, GRP, OCC_VM`*, where* `OCC_VM`$= occ_v(g)$

Such alerting system helps to catch VMs that are clustered into an infrequent group.



(a) (b)

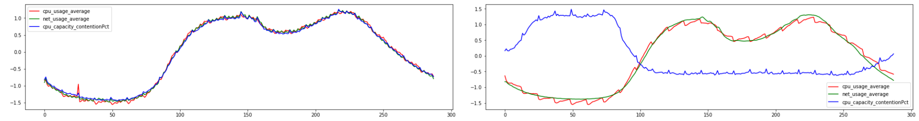
Fig. 4: (a) INFRA resource consumption clusters identified with the multi-metric analysis. The dark blue, green and light blue curves in each plot correspond to the `cpu|usage_average`, `net|usage_average` and `cpu|capacity_contentionPct` vROps metrics, respectively. (b) SOM grid showing the percentage of training samples captured by each neuron.

5 Experimental Results

In this section, we provide an overview of the results that can be obtained using the approach proposed in Section 4, that partially extend what has been already presented in our previous work [6]. For the analysis, we have relied on the experience of domain experts and focused our attention over a limited set of metrics that are considered the most relevant in this context, i.e., the ones related to the computational, networking and storage activity of VMs and VNFs of interest. Specifically, in the following, we highlight results obtained analyzing the following vROps metrics: `cpu|capacity_contentionPct`, `cpu|usage_average`, `net|usage_average`.

5.1 Multi-metric Analysis

The plots reported in Figure 4 are examples of the results that can be obtained through the multi-metric SOM-based analysis presented in Section 4, applied over a month worth of system-level (INFRA) metrics, using the normalized strategy. The trained SOM network is visually represented in terms of the weights of its neurons. Indeed, each subplot reports the VMs daily behavior that the specific neuron specialized into. In order to simplify the representation, the weight vectors – jointly computed over the three metrics `cpu|usage_average`, `net|usage_average` and `cpu|capacity_contentionPct` – are overlapped but in different colors. For instance, one of the most recurrent patterns, occurring in 17.46% of the observations and depicted in Figure 5a, is the one identified by the top-right neuron. Because of the standard data normalization performed during the preprocessing phase to discard the magnitude information in favor of enhancing the behavioral information of the input samples, the values on the



(a)

(b)

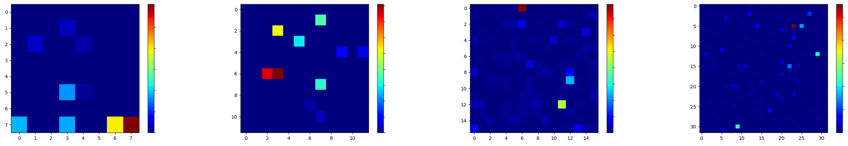
Fig. 5: (a) The most recurrent VM cluster of Figure 4a and (b) a singular VM pattern captured by the bottom-right neuron of Figure 4a (both appeared in [6]).

Y axis can be negative. This means that VMs have been clustered based on the joint shape of their daily resource consumption patterns, not their absolute values. Notice that in this example we can observe a quite suspect output, since the `cpu|capacity_contentionPct` figure follows closely the daily traffic pattern on the involved VMs. In a normal condition of a healthy system, i.e., when VMs are provided with appropriate computational resources, we would have expected this metric to stay close to zero, or at least experience a slight increase only during the peak hours. A significantly different pattern is the one reported in Figure 5b, corresponding to the bottom-right neuron in Figure 4a. Such behavior represents the 8.27% of the observed daily patterns in the time period under analysis. As evident from the picture, there is a higher CPU contention during night, when the VM has lower traffic, than during the day.

An additional remark regarding the possible presence of anomalies can be done considering the fact that the VMs included in the analysis are guaranteed to have the same role in the corresponding VNFs, i.e., they manage traffic in load sharing-mode. While it was expected to obtain an identical output for all of them, the SOM-based analysis has pointed out that a subset of such VMs exhibits daily patterns very different to the expected ones instead. This could be interpreted by human operators as a warning, that requires further monitoring and analysis of the involved components of the infrastructure. In addition, it is worth noticing that asynchronous changes among the metrics included in such analysis could be indications of anomalous behavior of the NFV environment, and not necessarily of the VNF itself.

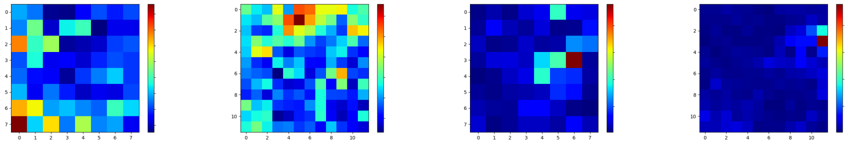
Table 1: The hyper-parameters values used for grid search (appeared in [6]).

Hyper-parameter	**Space**
SOM dimensions	8×8, 12×12, 16×16, 24×24, 32×32, 48×48
learning rate	$0.1, 0.2, \dots, 0.9, 1.0$
neighborhood radius (σ)	$0.1, 0.2, \dots, 0.9, 1.0$
epochs	$5, 10, 20$



(a) (b) (c) (d)

Fig. 6: SOMs with low σ values: (a) 8×8, σ: 0.1, lr: 0.2; (b) 12×12, σ: 0.1, lr: 0.2; (c) 16×16, σ: 0.1, lr: 0.9; (d) 32×32, σ: 0.1, lr: 0.8. For confidentiality reasons, the scale has been omitted (appeared in [6]).



(a) (b) (c) (d)

Fig. 7: SOMs with high σ values: (a) 8×8, σ: 0.6, lr: 0.2; (b) 12×12, σ: 0.6, lr: 0.2; (c) 8×8, σ: 0.6, lr: 0.9; (d) 12×12, σ: 0.6, lr: 0.9. For confidentiality reasons, the scale has been omitted (appeared in [6]).

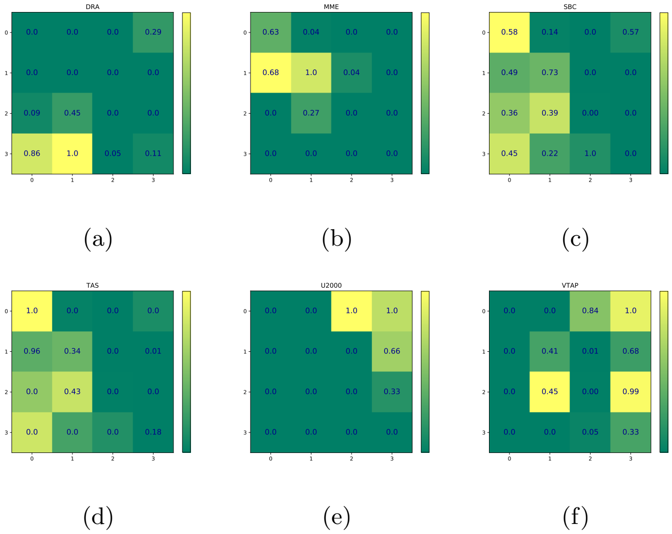
5.2 Hyper-parameters Grid Search

As mentioned in Section 4.1, different hyper-parameters lead to very different clusters after training. An extensive grid search has been conducted over the search space summarized in Table 1. A total of 1600 different configurations has been tested monitoring quantization error and readability of results. Figure 6 shows the effect of using a low σ value (0.1) in different map sizes. Using a low σ with a low learning rate gives the worst results with very few BMUs that capture more than 95% of data, resulting in higher quantization errors.

SOM maps greater than 12×12 require very high σ (> 0.8) and very low learning rate (< 0.3) in order to have low quantization errors, but in these cases the results tend to become unreadable due to the fact that too many neurons specialize on similar patterns. In Figure 7, the SOM maps reported in Figures 7a and 7b are trained using high σ and low learning rate, while the ones reported in Figures 7c and 7d are trained using high σ and high learning rate. Therefore, for our analysis the best combination of hyper-parameters are high values of σ (> 0.6) and low values of learning rate (< 0.6) with results that are better both in terms of quantization error and readability.

5.3 Per-VNF Analysis

Another interesting characterization we could perform applying the SOM-based analysis, is a study of how different VNFs behave in terms of their daily resource



(a) (b) (c)

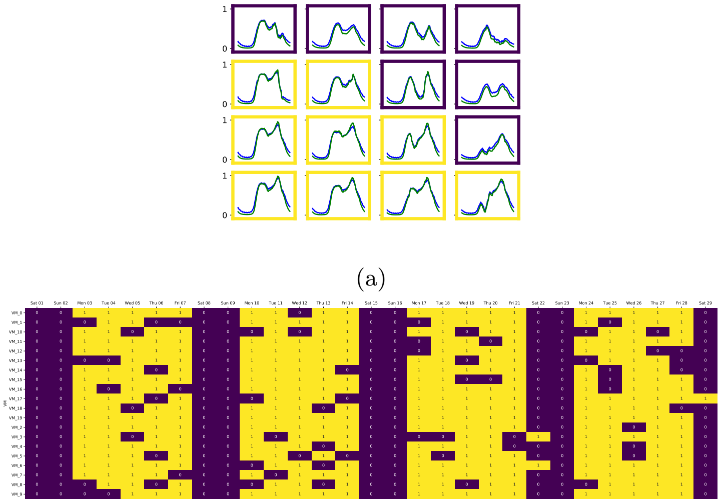
(d) (e) (f)

Fig. 8: SOM clusters and corresponding per-VNF hitmaps. For confidentiality reasons, the total number of hits in the hitmap cells has been rescaled to 1 (appeared in [6]).

consumption patterns. In this case, we produced hitmaps highlighting how many daily patterns of VMs of each given VNF map onto each SOM neuron. The result can be visualized as in Figure 8. For example, by comparing such plots with the corresponding map reporting the captured behaviors (like the one in Figure 4a, even though, in this case, the two figures are derived from different subsets of the available data), one can discover that both the SBC and the TAS VNFs have mostly the usual *"nightly/daily"* pattern, characterized by a low workload over nightly hours and a high workload over daily hours, with peaks around noon and 6pm. On the other hand, the DRA VNF exhibits the classical nightly/daily pattern for the `cpu|capacity_contentionPct` metric, and periodic peaks every 30 minutes for the other two metrics. Moreover, a consistent number of VTAP VMs are characterized by hourly periodic peaks.

5.4 Hierarchical Grouping

In this section, we report two examples of grouping/clustering technique described in Section 4.3, starting from another month of data, with respect to the experiments shown above. In the first example, we obtained the trained SOM whose weights are presented in Figure 9a. By applying the distance-based grouping, with a group-distance threshold of 0.007, we obtained the clustering shown in the same figure, where neurons belonging to the same group have the same border color. Moreover, to facilitate a visual inspection of the behavior of each VM during the month, we produced a calendar view of the VMs, as shown in

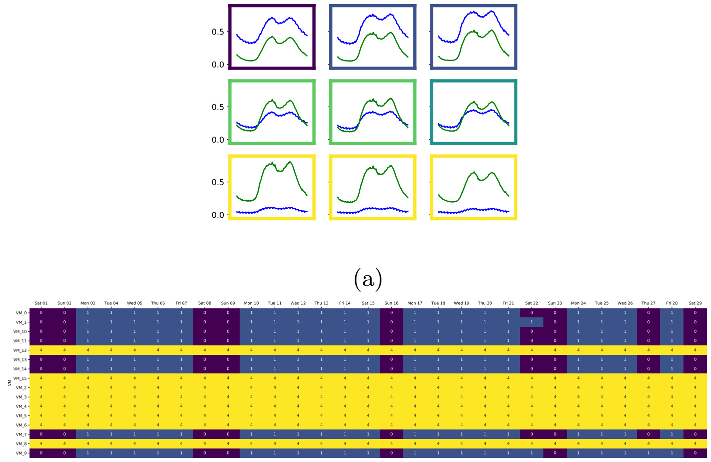


(a)

	Sat 01	Sun 02	Mon 03	Tue 04	Wed 05	Thu 06	Fri 07	Sat 08	Sun 09	Mon 10	Tue 11	Wed 12	Thu 13	Fri 14	Sat 15	Sun 16	Mon 17	Tue 18	Wed 19	Thu 20	Fri 21	Sat 22	Sun 23	Mon 24	Tue 25	Wed 26	Thu 27	Fri 28	Sat 29
VM_0	0	0	1	1	1	1	1	0	0	1	1	0	1	1	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0
VM_1	0	0	0	1	1	0	0	0	0	1	1	1	1	1	0	0	0	1	1	1	1	0	0	1	0	1	1	1	0
VM_10	0	0	1	1	0	1	1	0	0	0	1	0	1	1	0	0	1	1	0	1	1	0	0	0	1	1	0	1	0
VM_11	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0	0	0	1	1	0	1	0	0	1	1	1	1	1	0
VM_12	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0	0	0	1	1	1	1	0	0	1	1	1	0	0	0
VM_13	0	0	0	0	1	1	1	0	0	1	1	1	1	1	0	0	1	1	0	1	1	0	0	0	1	1	1	0	0
VM_14	0	0	1	1	1	0	1	0	0	1	1	1	1	0	0	0	1	1	1	1	1	0	0	1	0	1	1	0	0
VM_15	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0	0	1	1	0	0	1	0	0	1	0	1	1	1	0
VM_16	0	0	1	0	1	1	0	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0	0	1	0	1	1	1	0
VM_17	0	0	1	1	1	0	1	0	0	0	1	1	1	0	0	0	1	1	1	1	1	0	0	1	1	1	1	1	1
VM_18	0	0	1	1	0	1	1	0	0	1	1	1	0	1	0	0	1	1	1	1	1	0	0	1	1	1	1	0	0
VM_19	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0
VM_2	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0	0	1	1	0	1	1	0
VM_3	0	0	1	1	0	1	1	0	0	1	0	1	1	1	0	0	0	0	1	1	0	1	0	1	1	1	1	1	0
VM_4	0	0	1	1	1	1	1	0	0	1	1	1	0	1	0	0	1	1	1	1	0	0	0	1	1	0	1	1	0
VM_5	0	0	1	1	1	0	1	0	0	1	1	0	1	0	0	0	1	0	1	1	1	0	0	1	1	0	1	1	0
VM_6	0	0	1	1	1	1	1	0	0	0	1	1	0	1	0	0	1	1	1	1	1	1	0	1	1	1	1	1	0
VM_7	0	0	1	1	1	1	0	0	0	1	0	1	1	1	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0
VM_8	0	0	0	1	1	0	1	0	0	0	1	1	0	1	0	0	1	1	0	1	1	0	0	0	1	1	1	1	0
VM_9	0	0	0	0	1	1	1	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0

(b)

Fig. 9: (a) Distance-based grouping applied to a square SOM grid, 4 neurons per side. (b) VMs exhibiting common behaviors.



(a)

	Sat 01	Sun 02	Mon 03	Tue 04	Wed 05	Thu 06	Fri 07	Sat 08	Sun 09	Mon 10	Tue 11	Wed 12	Thu 13	Fri 14	Sat 15	Sun 16	Mon 17	Tue 18	Wed 19	Thu 20	Fri 21	Sat 22	Sun 23	Mon 24	Tue 25	Wed 26	Thu 27	Fri 28	Sat 29
VM_0	0	0	1	1	1	1	1	0	0	1	1	1	1	1	1	0	1	1	1	1	1	0	0	1	1	1	0	1	0
VM_1	0	0	1	1	1	1	1	0	0	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1	1	0	1	0
VM_10	0	0	1	1	1	1	1	0	0	1	1	1	1	1	1	0	1	1	1	1	1	0	0	1	1	1	0	1	0
VM_11	0	0	1	1	1	1	1	0	0	1	1	1	1	1	1	0	1	1	1	1	1	0	0	1	1	1	0	1	0
VM_12	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
VM_13	0	0	1	1	1	1	1	0	0	1	1	1	1	1	1	0	1	1	1	1	1	0	0	1	1	1	0	1	0
VM_14	0	0	1	1	1	1	1	0	0	1	1	1	1	1	1	0	1	1	1	1	1	0	0	1	1	1	0	1	0
VM_15	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
VM_2	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
VM_3	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
VM_4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
VM_5	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
VM_6	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
VM_7	0	0	1	1	1	1	1	0	0	1	1	1	1	1	1	0	1	1	1	1	1	0	0	1	1	1	0	1	0
VM_8	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
VM_9	0	0	1	1	1	1	1	0	0	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1	1	1	1	0

(b)

Fig. 10: (a) Distance-based grouping applied to a square SOM grid, 3 neurons per side. (b) VMs exhibiting anomalous behaviors.

Figure 9b, by associating to each couple (VM, DAY) the group of the BMU in the SOM grid. For instance, in this case we can notice a very common behavior: the majority of group changes take place during the week-end. The second example shows how the two outputs could be jointly used by a system operator to visually detect anomalies in the behavior of VMs. The grouped SOM grid in Figure 10a highlights three main (i.e, more frequent) groups. In particular, the yellow group contains all the neurons within an almost flat VM metric. By inspecting the calendar in Figure 10b, it is evident that these behaviors are associated to those VMs that have an anomalous *constant* course, without any variations during the week-end.

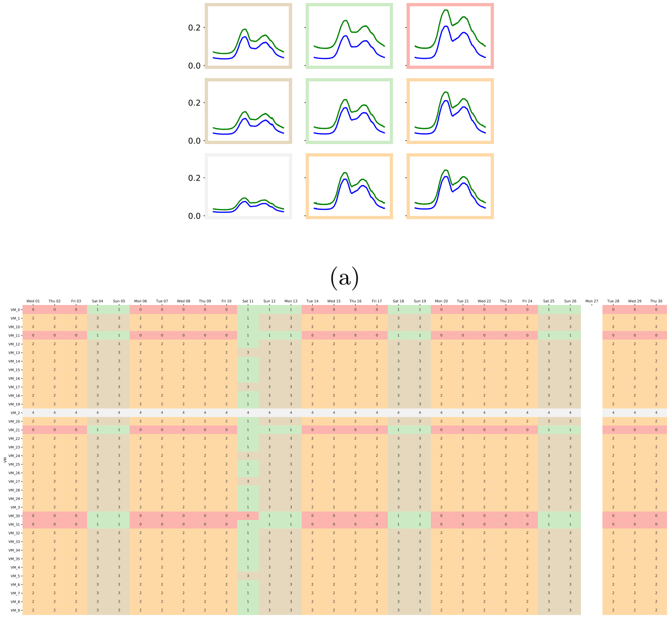
5.5 Alerting

In this section, we provide some examples of output of the two main kind of alerting systems. All the alerting systems are applied to the behaviors captured by the SOM grid in Figure 11a. Notice the presence of two groups with a high working-level (red, orange); two groups with a low working-level (brown, green); one group with a unique almost flat neuron (gray). Figure 11b reports the behavioral evolution of the VMs whose data have been used to conduct this experiment. The reference time period is April 2020 and, in particular, on April 13th (Easter Monday) many VMs change behavior, passing from their usual high working-level group to a low working-level group.

Calendar-like Alerting Systems Figure 12a shows the *Strong Alerting System* defined in Equation (SAS), with $m = 3$ and $p = 7$, where dark green cells stand for a raised alert. Since the method compares the groups in same week-days and raises an alert if at least one change occurs, we can see that many alerts have been raised (some of them are obviously false positives). In contrast, Figure 12b shows the alerts that have been raised with the *Weak Alert System* defined in Equation (WAS), with $p = 7$ and $m = 3$. Since the method raises an alert if a group appears only once in same week-days, we can see that only a few behaviors raise an alert. The output from the *Fuzzy Alerting System* defined in Equation (ZAS), with $p = 7$, is shown in Figure 12c. The higher the value (i.e., the darker the color), the higher the probability of an alert being significant. As expected, many of the *false-positives* reported by the *Strong Alerting System*, and not by the *Weak* one, are associated to a low value.

6 Conclusions

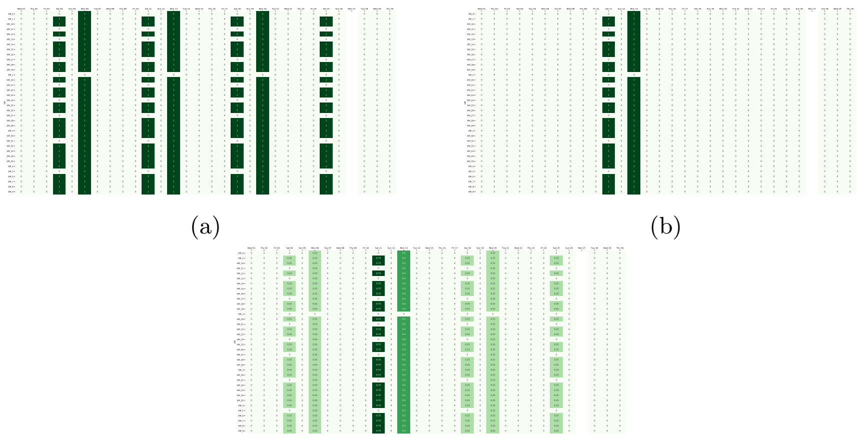
In this work, we focused on the problem of analysis and classification of the behavioral patterns of VM metrics in a NFV data center. We described the technique we realized, based on self-organizing maps, that is being used across the data centers of the Vodafone network operator. We described some results we obtained from its application, highlighting the capability of our technique to identify interesting points in space and time (i.e., precise VMs and hosts



(a)

(b)

Fig. 11: (a) A SOM grid with neurons grouped in 5 behaviors. (b) Calendar view of the set of VMs involved in the analysis.



(a)

(b)

(c)

Fig. 12: (a) Strong Alerting System. (b) Weak Alerting System. (c) Fuzzy Alert System.

within the infrastructure, and precise days within the analyzed time range) with potentially anomalous behaviors, thus deserving further attention and investigations by data center operators. Also, we detailed a clustering technique applied over a trained SOM codebook in order to mitigate the problem of neuron over-representation, and an alerting system built atop the SOM-based clustering, improving the anomaly detection pipeline effectively reducing the number of false positives.

In our experimentation, we identified a variety of open questions that still need additional investigations. First, the proposed technique has a number of hyper-parameters (SOM grid size and parameters, and various thresholds as described in Section 4) that have to be decided. A grid search can be used for such purpose, but it requires a non-negligible processing time as possible configurations can easily grow in the range of tens or hundreds. In order to select the best SOM hyper-parameters, the various analysis runs should be compared with one another using an automated and quantitative assessment method. This cannot be simply done based on the SOM quantization error, as it would decrease increasing the SOM size, driving the choice towards excessively large networks. For example, we plan to use the average silhouette width to such purpose [29]. Finally, another promising path we plan to explore is the one to combine our approach with the use of Deep Learning (DL) for time-series classification [18,19,21]. An interesting approach could be using the SOM to produce a more compact and discrete representation of a time-series autoencoder, as explained in [9].

References

1. Van den Berg, F.D., Kok, P., Yang, H., Aarnts, M., Meilland, P., Kebe, T., Stolzenberg, M., Krix, D., Zhu, W., Peyton, A., et al.: Product uniformity control-a research collaboration of european steel industries to non-destructive evaluation of microstructure and mechanical properties. In: Electromagnetic Non-Destructive Evaluation (XXI). 6 September 2017 through 8 September 2017. pp. 120–129 (2018)
2. Bertero, C., Roy, M., Sauvanaud, C., Tredan, G.: Experience Report: Log Mining Using Natural Language Processing and Application to Anomaly Detection. In: 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE). pp. 351–360. IEEE (oct 2017)
3. Buczak, A.L., Guven, E.: A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Communications Surveys & Tutorials **18**(2), 1153–1176 (2015)
4. Canetta, L., Cheikhrouhou, N., Glardon, R.: Applying two-stage SOM-based clustering approaches to industrial data analysis. Production Planning & Control **16**(8), 774–784 (2005)
5. Chen, D.R., Chang, R.F., Huang, Y.L.: Breast cancer diagnosis using self-organizing map for sonography. Ultrasound in medicine & biology **26**(3), 405–411 (2000)
6. Cucinotta, T., Lanciano, G., Ritacco, A., Vannucci, M., Artale, A., Barata, J., Sposato, E., Basili, L.: Behavioral analysis for Virtualized Network Functions: a SOM-based approach. In: Proceedings of the 10th International Conference on Cloud Computing and Services Science. Prague, Czech Republic (May 2020)

7. Díaz, I., Domínguez, M., Cuadrado, A.A., Fuertes, J.J.: A new approach to exploratory analysis of system dynamics using som. applications to industrial processes. Expert Systems with Applications **34**(4), 2953 – 2965 (2008)
8. Farshchi, M., Schneider, J.G., Weber, I., Grundy, J.: Metric selection and anomaly detection for cloud operations using log and metric correlation analysis. Journal of Systems and Software **137**, 531–549 (mar 2018)
9. Fortuin, V., Hüser, M., Locatello, F., Strathmann, H., Rätsch, G.: SOM-VAE: Interpretable Discrete Representation Learning on Time Series. In: International Conference on Learning Representations (2019)
10. Frey, C.W.: Monitoring of complex industrial processes based on self-organizing maps and watershed transformations. In: 2012 IEEE International Conference on Industrial Technology. pp. 1041–1046. IEEE (2012)
11. Fuertes, J.J., Domìnguez, M., Reguera, P., Prada, M.A., Dìaz, I., Cuadrado, A.A.: Visual dynamic model based on self-organizing maps for supervision and fault detection in industrial processes. Engineering Applications of Artificial Intelligence **23**(1), 8–17 (2010)
12. Gulenko, A., Wallschlager, M., Schmidt, F., Kao, O., Liu, F.: Evaluating machine learning algorithms for anomaly detection in clouds. In: Proc. IEEE International Conference on Big Data. pp. 2716–2721. IEEE (dec 2016)
13. Gulenko, A., Wallschläger, M., Schmidt, F., Kao, O., Liu, F.: A system architecture for real-time anomaly detection in large-scale nfv systems. Procedia Computer Science **94**, 491–496 (2016), the 11th International Conference on Future Networks and Communications (FNC 2016) / The 13th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2016) / Affiliated Workshops
14. Gupta, L., Samaka, M., Jain, R., Erbad, A., Bhamare, D., Chan, H.A.: Fault and performance management in multi-cloud based NFV using shallow and deep predictive structures. Journal of Reliable Intelligent Environments **3**(4) (jul 2017)
15. Harris, T.: A kohonen som based, machine health monitoring system which enables diagnosis of faults not seen in the training set. In: Proceedings of 1993 International Conference on Neural Networks. vol. 1, pp. 947–950. IEEE, Nagoya, Japan (1993)
16. Hawilo, H., Shami, A., Mirahmadi, M., Asal, R.: NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC). IEEE Network **28**(6), 18–26 (nov 2014)
17. Haykin, S.: Neural Networks: A Comprehensive Foundation (3rd Edition). Prentice-Hall, Inc., USA (2007)
18. Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A.: Deep learning for time series classification: a review. Data Mining and Knowledge Discovery **33**(4), 917–963 (jul 2019)
19. Kashiparekh, K., Narwariya, J., Malhotra, P., Vig, L., Shroff, G.: Convtimenet: A pre-trained deep convolutional neural network for time series classification. In: International Joint Conference on Neural Networks, IJCNN 2019 Budapest, Hungary, July 14-19, 2019. pp. 1–8. IEEE (2019)
20. Le, L., Sinh, D., Lin, B.P., Tung, L.: Applying Big Data, Machine Learning, and SDN/NFV to 5G Traffic Clustering, Forecasting, and Management. In: 4th IEEE Conference on Network Softwarization and Workshops. pp. 168–176 (June 2018)
21. Malhotra, P., TV, V., Vig, L., Agarwal, P., Shroff, G.: TimeNet: Pre-trained deep recurrent neural network for time series classification. In: 25th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, 2017, Bruges, Belgium (2017)

22. Malini, N., Pushpa, M.: Analysis on credit card fraud identification techniques based on knn and outlier detection. In: 2017 Third International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB). pp. 255–258 (Feb 2017)
23. Mancini, R., Ritacco, A., Lanciano, G., Cucinotta, T.: XPySom: High-Performance Self-Organizing Maps. In: IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). Porto, Portugal (2020)
24. Miyazawa, M., Hayashi, M., Stadler, R.: vnmf: Distributed fault detection using clustering approach for network function virtualization. In: IFIP/IEEE International Symposium on Integrated Network Management. pp. 640–645 (May 2015)
25. NFV Industry Specif. Group: Network Functions Virtualisation. Introductory White Paper (2012)
26. Niwa, T., Miyazawa, M., Hayashi, M., Stadler, R.: Universal fault detection for nfv using som-based clustering. In: 2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS). pp. 315–320 (Aug 2015)
27. Ostberg, P.O., Byrne, J., Casari, P., Eardley, P., Anta, A.F., Forsman, J., Kennedy, J., Le Duc, T., Marino, M.N., Loomba, R., Pena, M.A.L., Veiga, J.L., Lynn, T., Mancuso, V., Svorobej, S., Torneus, A., Wesner, S., Willis, P., Domaschka, J.: Reliable capacity provisioning for distributed cloud/edge/fog computing applications. In: EuCNC 2017 - European Conference on Networks and Communications. pp. 1–6. IEEE (jun 2017)
28. Pitakrat, T., Okanović, D., van Hoorn, A., Grunske, L.: Hora: Architecture-aware online failure prediction. Journal of Systems and Software **137**, 669–685 (mar 2018)
29. Rousseeuw, P.J.: Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. J. of Computational and Applied Mathematics **20**, 53–65 (1987)
30. Samrin, R., Vasumathi, D.: Review on anomaly based network intrusion detection system. In: 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques. pp. 141–147 (Dec 2017)
31. Sauvanaud, C., Kaâniche, M., Kanoun, K., Lazri, K., Da Silva Silvestre, G.: Anomaly detection and diagnosis for cloud services: Practical experiments and lessons learned. Journal of Systems and Software **139**, 84–106 (may 2018)
32. Van Gassen, S., Callebaut, B., Van Helden, M.J., Lambrecht, B.N., Demeester, P., Dhaene, T., Saeys, Y.: Flowsom: Using self-organizing maps for visualization and interpretation of cytometry data. Cytometry Part A **87**(7), 636–645 (2015)
33. Vannucci, M., Colla, V.: Novel classification method for sensitive problems and uneven datasets based on neural networks and fuzzy logic. Applied Soft Computing **11**(2), 2383–2390 (2011)
34. Wallschläger, M., Gulenko, A., Schmidt, F., Kao, O., Liu, F.: Automated Anomaly Detection in Virtualized Services Using Deep Packet Inspection. In: Procedia Computer Science. vol. 110, pp. 510–515. Elsevier (jan 2017)
35. Watanabe, Y., Otsuka, H., Sonoda, M., Kikuchi, S., Matsumoto, Y.: Online failure prediction in cloud datacenters by real-time message pattern learning. In: CloudCom 2012 - Proceedings: 2012 4th IEEE International Conference on Cloud Computing Technology and Science. pp. 504–511. IEEE (dec 2012)
36. Wittek, P., Gao, S.C., Lim, I.S., Zhao, L.: somoclu : An Efficient Parallel Library for Self-Organizing Maps. Journal of Statistical Software **78**(9) (jun 2017)