An RBM Anomaly Detector for the Cloud

Cristina Monni[1], Mauro Pezzè[1,2] and Gaetano Prisco[3]
[1]USI Università della Svizzera italiana
[2]Università degli studi di Milano Bicocca
[3]Università degli studi di Salerno
{cristina.monni, mauro.pezze}@usi.ch, g.prisco21@studenti.unisa.it

Abstract—**Failures are unavoidable in complex software systems, and the intrinsic characteristics of cloud systems amplify the problem. Predicting failures before their occurrence by detecting anomalies in system metrics is a viable solution to enable failure preventing or mitigating actions. The most promising approaches for predicting failures exploit statistical analysis or machine learning to reveal anomalies and their correlation with possible failures. Statistical analysis approaches result in far too many false positives, which severely hinder their practical applicability, while accurate machine learning approaches need extensive training with seeded faults, which is often impossible in operative cloud systems.**

In this paper, we propose *EmBeD*, *Energy-Based anomaly Detection in the cloud*, an approach to detect anomalies at runtime based on the free energy of a Restricted Boltzmann Machine (RBM) model. The free energy is a stochastic function that can be used to efficiently score anomalies for detecting outliers. *EmBeD* analyzes the system behavior from raw metric data, does not require extensive training with seeded faults, and classifies the relation of anomalous behaviors with future failures with very few false positives. The experimental results presented in this paper confirm that *EmBeD* can precisely predict failure-prone behavior without training with seeded faults, thus overcoming the main limitations of current approaches.

Index Terms—**anomaly detection, failure prediction, cloud reliability, machine learning**

I. Introduction

Runtime failures are unavoidable in complex software systems, due to the complexity of the systems, the constraints of development environments and the heterogeneity of the production environments [1]. The intrinsic characteristics of cloud systems amplify the problem and reduce the effectiveness of classic testing approaches. Several factors increase the frequency of failures, and harden the problem of locating faults in the cloud: (i) the distributed and heterogeneous ownership and management of cloud platforms and applications, (ii) the dynamic allocation of resources and the adoption of commodity hardware, and (iii) the growing complexity of applications and system-environment interactions. These factors reduce the effectiveness of classic testing and analysis approaches that cannot rely on exact replica of the systems on in-house development platform and may not be able to access sometimes relevant components of the system [1].

When some executions trigger one or more faults, the system may deviate from the correct state and incur in erroneous states that may lead to system failures, that is, incorrect externally visible behaviors. Erroneous states often reflect in many anomalous values for metrics that are commonly collected or easily collectable in software systems.

Modern failure predictors monitor metrics collected at some nodes as *Key Performance Indicators* (KPIs), and analyse series of KPIs, to identify out-of-norm values, also called *anomalies*, and they infer the presence of erroneous states from the identified anomalies. Anomalous values of single KPIs (*single anomalies*) reflect specific changes in usually correct system executions, and are quite common in complex systems, while anomalies that collectively involve several KPIs over time (*collective anomalies*) are less usual and may reflect either correct behaviors that extensively deviate from the normal execution or erroneous system states [2, 3, 4, 5].

The main challenge of failure detectors based on anomalies is to distinguish *failure-prone collective anomalies*, that is, anomalies that would lead to failures if not properly fixed, from *benign collective anomalies*, that is, anomalies that occur for unusual but correct executions, and do not raise failures [6].

Data-driven approaches rely on the analysis of data collected with black-box monitoring. They can identify previously unseen behaviors, and are particularly effective when dealing with systems characterized by complex architectures, concurrency and multitenancy, such as cloud systems [7].

Data-driven approaches usually retrieve models of normal behavior through data analytics and/or machine learning. Data analytics approaches are affected by many false positives, while the accuracy and precision of machine learning approaches depend on the data used for training the models [7].

Unsupervised machine learning approaches train the models with data collected from correct executions, which are largely available but provide limited information about the behavior of the system in the presence of failures. Unsupervised models produce many false alarms that hinder their efficacy [8], and often suffer from low precision [9], limited generalizability [10] or high overhead [9, 11, 8, 12].

Supervised machine learning approaches train the models with data collected from both normal and failing executions [6, 2, 13, 14, 15, 16, 17]. Data from failing executions may be obtained from failures experienced in production environments, which are fairly rare events in mature systems, or from artificially seeded faults, which are often not allowed in 24/7 business-critical cloud systems. Supervised approaches can accurately reveal only anomalies relative to the types of behaviors encoded in the seeded faults.

In this paper, we propose *EmBeD*, *Energy-Based anomaly*

Detection in the cloud, an approach that (i) reveals failure-prone anomalies in cloud systems at runtime, (ii) is data-driven and is thus independent from the target system architecture, (iii) exploits machine learning and thus adapts to the continuous changes in the configuration, and (iv) trains models in an unsupervised fashion and thus avoids relying on errors seeded in production environment. *EmBeD* is as precise and accurate as supervised machine learning approaches, and overcomes the limitations of current unsupervised approaches by relying on a lightweight model, like data analytics approaches.

EmBeD is ground on a recent work [18] where we investigated the analogies between failure-prone anomalies in complex cloud systems and non-trivial behaviors of complex physical and network systems, like the spontaneous magnetization of metals [19, 20], and the emergence of epidemic thresholds in airport networks [21]. The non-trivial behaviors of complex physical and network systems can be predicted and analyzed with a stochastic approach by computing the *Gibbs free energy* [19, 20], a function of random variables that represent the target system over time. Based on the results of our feasibility study [18], *EmBeD* computes the Gibbs free energy as a function of the KPIs of a cloud system by training a Restricted Boltzmann Machine (RBM) and measures the anomaly level of the status of complex cloud systems in terms of free energy. Our experimental results indicate a strong relationship between anomalies indicated by the free energy and the presence of failure-prone collective anomalies.

The paper is organized as follows. Section II describes the core characteristics of *EmBeD*, outlines the essential elements of RBMs that are required to understand *EmBeD*, and discusses the the main logical components of *EmBeD*. Section III describes the experimental setup that we used to evaluate *EmBeD*. Section IV discusses the experimental results that confirm the effectiveness and efficiency of *EmBeD*. Section V overviews the main approaches for detecting and predicting failure-prone behavior in cloud systems, and surveys the main application of Restricted Boltzmann Machines (RBMs) in other domains. Section VI outlines the main contributions of this work and suggests a research roadmap.

II. *EmBeD*

EmBeD identifies failure-prone anomalies in cloud systems from significant deviations of the free energy of the system. It computes the reference values of the free energy during a training phase by executing the system under normal conditions, and calculates the free energy when the system operates regularly, to identify values that significantly differ from the reference free energy. *EmBeD* trains the system without information about the presence of failures in the training dataset, which means that it does need seeded faults, computes the free energy from series of KPI values monitored during a short training time, thus without long training sessions, calculates the free energy in linear time, thus promptly reporting anomalies, and executes the analysis on nodes external to the cloud, thus with negligible interference with the system operations.

A. Anomalies and failures

EmBeD builds, updates and analyzes the free energy from series of *KPI* values observed over time at different abstraction levels and from different resources. Example of KPIs are 'CPU usage' at a server level, 'number of bytes sent' or 'received' at the Virtual Machine (VM) level, 'memory used' at the application level. Cloud systems are characterized by several thousands of KPIs, and failure-prone behaviors usually affect many KPIs, due to the distributed architecture of the system and the dynamic interaction among components. Failure-prone behaviors are not reducible to the single components, but involve the analysis of thousands of KPI values, and of their mutual relations over time.

Recent work [22, 23] shows that KPIs mutually interact, and that the number of anomalous KPIs increases over time when the system is in erroneous states, thus suggesting that the interaction among KPIs may be a symptom of the spread of errors in the system. *EmBeD* works under the assumption that anomalous KPIs evolve and often increase over time [22, 23] when the system remains in error states while maturing a failure, thus the status of the system strongly relates to the *collective* interactions among KPIs. However, KPI time series need accurate analysis and filtering, since they may not necessarily be related to erroneous states in the system [22]. Modeling the interactions among KPIs is computationally hard when dealing with thousands of KPIs and hundreds of thousands of interactions [24], and becomes impractical for proactively detecting failures. *EmBeD* efficiently deals with the large amount of data that derive from the interactions of KPIs by means of a stochastic approach. It classifies the system state as 'normal' or 'anomalous' with the Restricted Boltzmann Machine (RBM), a stochastic neural network whose nodes and weights model KPIs and their interactions, respectively. RBMs compute the *free energy* from the probability distribution of the system configuration, that reflects the collective behavior of KPIs and is thus an indicator of the status of the system (normal/anomalous). The main advantage of RBMs with respect to current anomaly detection approaches is that they are generative models [25, 26], which means that they learn how to reproduce configurations with the same relevant patterns of the training set, thus coping with noise and mixed data, that typically affect KPI streams.

As shown in Figure 1, *EmBeD* uses a RBM to compute the free energy of the KPIs collected from the system both during model training and for detecting anomalies. KPIs are metrics collected from different resources, such as *Bytes send* in the VM *bono* or *Loss rate* in the VM *homer*. In the *training* phase, *EmBeD* builds a *parametric baseline model* from the free energy of the system KPIs under normal operation conditions (*model builder*), and sets the parameters of the baseline model later used to discriminate anomalies (*parameter selector*). In the *detection* phase, *EmBeD* compares the free energy in the current execution window to detect anomalies with respect to the baseline model (*anomaly detector*).

In the following, we introduce the RBM, the core compo-

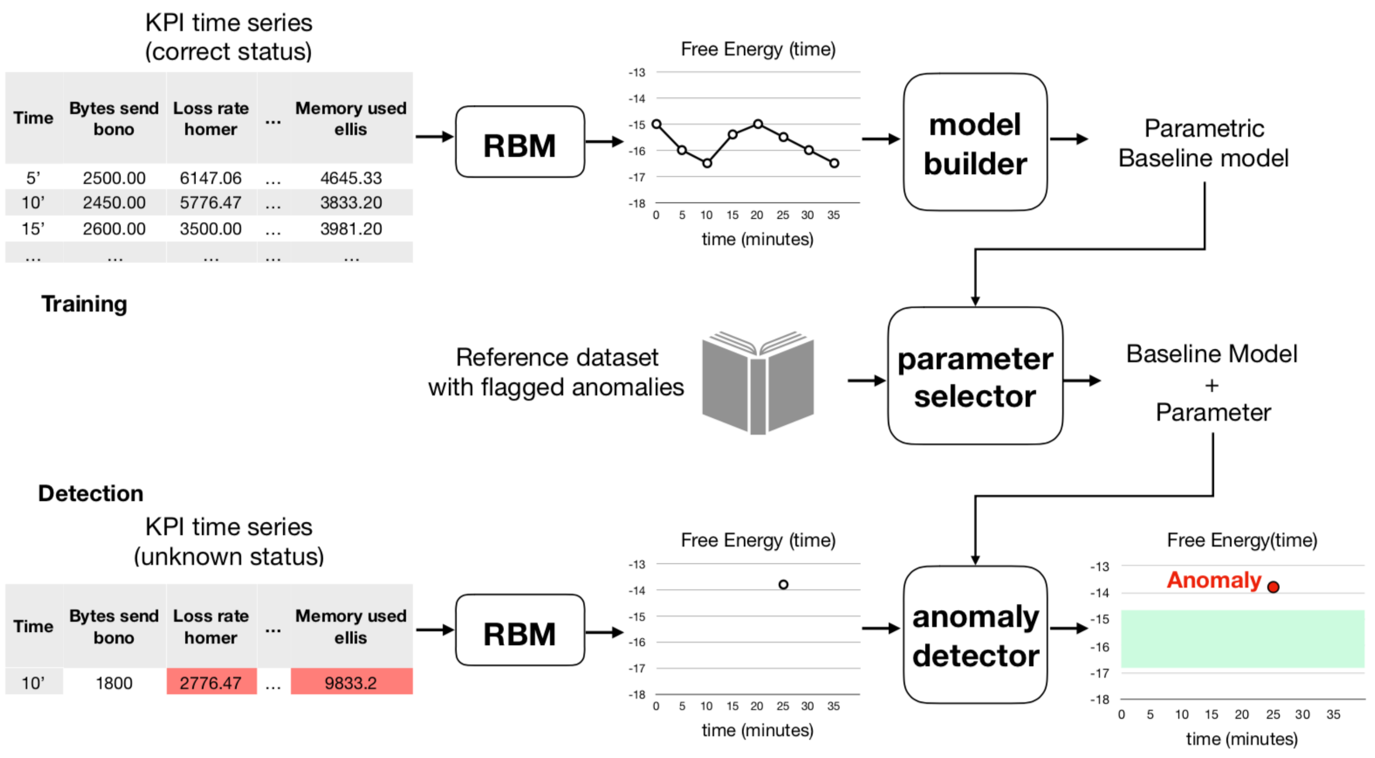

Fig. 1. Model building, parameter selection and anomaly detection phases of *EmBeD*

nent of *EmBeD*. We describe how *EmBeD* trains a RBM to compute both the free energy of normal behavior and its related baseline model (*model builder*). We discuss how *EmBeD* selects the best baseline models and sets the parameters by evaluating the models with a dataset with flagged anomalies (*parameter selector*). We show how *EmBeD* detects anomalies by comparing the free energy of streaming data with the baseline model (*anomaly detector*).

B. RBM

In this Section we overview the concepts of the RBM strictly required to understand how *EmBeD* builds the baseline model by training a RBM. The interested readers can find a detailed survey on RBMs in the excellent paper by Fischer et al. [27]. We use the RBM, since both Le Roux and Bengio [28] and later Montufar and Nihat [29] have shown that the RBM computes a universal approximation of the probability distributions of stochastic variables, and is thus suitable to model KPIs in the form of time series.

As illustrated in Figure 2, a RBM is an undirected graph consisting of m visible nodes $\mathbf{v} \in [0,1]^m$ that represent observable data (in our case the input values of KPIs), and n hidden nodes $\mathbf{h} \in [0,1]^n$ that represent stochastic variables. When RBMs are used for selecting features, the number of hidden nodes n is smaller than the number of visible nodes m, up to one order of magnitude, to select a small set of relevant features [30]. In *EmBeD*, we use $n = m$, an amount of hidden nodes equal to the amount of input KPIs, to precisely identify anomalies. Weighted edges connect visible nodes and hidden

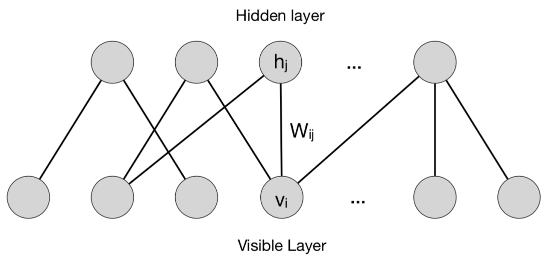

Fig. 2. Restricted Boltzmann Machine

nodes. Both the visible and the hidden layers are associated with a bias vector. The weights of the edges are conventionally represented with a matrix $\mathbf{W}$, and the bias vectors of the visible and hidden layers with vectors $\mathbf{a}$ and $\mathbf{b}$, respectively.

RBMs are bipartite Boltzmann machines [27], and can be interpreted as stochastic neural networks with symmetric connections across layers and no connections between nodes within the same layer. The restriction to intra-layer symmetric connections that characterises RBMs implies that all values h_j are conditionally independent from the configuration of visible nodes, and that all v_i are conditionally independent from the configuration of hidden nodes. Thus, an RBM can sample the conditional probabilities of all hidden nodes given the visible nodes with a parallel computation (and viceversa), with a substantial reduction of training costs [27].

A RBM computes the joint probability distribution of the input data with respect to the set of parameters $\theta = \{\mathbf{W}, \mathbf{a}, \mathbf{b}\}$

(the weights $\mathbf{W}$ of the edges and the biases $\mathbf{a}$ and $\mathbf{b}$ of the nodes) by associating a scalar energy function $E(\mathbf{v}, \mathbf{h})$ to each configuration [31]:

$$p_\theta(\mathbf{v}, \mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}} \tag{1}$$

$$E(\mathbf{v}, \mathbf{h}) = \mathbf{v}^T\mathbf{W}\mathbf{h} - \mathbf{b}^T\mathbf{h} - \mathbf{a}^T\mathbf{v} \tag{2}$$

The parameters $\theta = \{\mathbf{W}, \mathbf{a}, \mathbf{b}\}$ are initialized to uniformly distributed random values, and are iteratively updated, until reaching a reconstruction probability distribution $p_\theta(\mathbf{v}, \mathbf{h})$ that best approximates the distribution of the training data via stochastic gradient descent on the empirical negative log-likelihood of the training data. The reconstruction error quantifies the discrepancy between the values of reconstruction phase and the input values.

The RBM efficiently approximates the log-likelihood, which is an effective approximation of the distribution p, by computing the *Gibbs free energy* $G(\mathbf{v})$, which is a scalar function of the visible nodes:

$$G(\mathbf{v}) = -\log \sum_h e^{-E(\mathbf{v},\mathbf{h})} \tag{3}$$

The intuitive meaning of the free energy as an approximation of the status of the system can be grasped from thermodynamics, where it was originally defined as a potential that assumes its minimum value when the system reaches the *most likely configuration* (thermodynamic equilibrium). In the RBM the *most likely configuration* corresponds to the configuration of the visible units that best approximates the input, and is obtained by minimizing the free energy of the input during training.

In the *training phase*, the RBM does not require data labels, thus *EmBeD* builds the model of normal behavior by training the RBM in an unsupervised fashion, under the assumption that the training data contain a small percentage of anomalies (below 0.01%), which corresponds to a realistic scenario, since anomalies are usually rare events in mature cloud systems. Table I shows the structure of the training data, in a tabular form, where each row corresponds to the values of the KPIs collected at a given time sample, and thus each column corresponds to the values of a KPI in the form of time series. An input of the RBM corresponds to a row of the matrix in Table I. KPIs are pairs $(metric, resource)$: for example, the first KPI in Table I is the metric *Bytes send* in the resource named *bono*. The metrics and resources indicated in Table I are excerpted from the validation dataset described in Section III-B. Since *EmBeD* activates the RBM with a sigmoid function that requires the input data to be either boolean or normally distributed between 0 and 1, we normalize each time series as a normal distribution between 0 and 1.

TABLE I
EXAMPLE TRAINING SET AFTER NORMALIZATION

Time	*Bytes send* bono	*Loss rate* homer	...	*Memory used* ellis
08/09/2017 15:05	0.348	0.136	...	0.016
08/09/2017 15:10	0.447	0.164	...	0.015
...	...	...	...	...
08/09/2017 15:50	0.587	0.131	...	0.016

C. Model Builder

The *model builder* uses the free energy computed with the RBM from Equation (3) to build the baseline model for the anomaly detection. The *model builder* trains the RBM with KPI values monitored during normal executions, and computes the free energy that corresponds to the best approximation of the probability distribution of the training data. The computed free energy represents the distribution of normal behavior as a function of time with a standard deviation σ. Thus the baseline model of the free energy is a parametric function of σ. The *model builder* builds a *time-dependent* and a *constant* baseline model, the two models depend on both the standard deviation σ of the free energy and a multiplicative parameter $k \in \mathbb{N}$. The standard deviation σ statistically quantifies the variation range of the model, and the multiplicative parameter $k \in \mathbb{N}$ defines the width of the baseline related to the *tolerance* of the model. Small values of k result in a a small range of free energy values that indicate normal behavior, thus a low tolerance. Large values of k result in a larger set of free energy values that label normal behavior, thus a higher tolerance of the baseline model.

The *time-dependent* baseline model centers the interval on the trend line that best fits the free energy of the training set G_{train}:

$$trend\,line(G_{train}) \pm k\,\sigma(G_{train}) \tag{4}$$

The time-dependent baseline model follows the behavior of the free energy function over time, but is more sensitive to fluctuations in the system behavior.

The *constant* baseline interval is centered on the average of the free energy of the training set:

$$average(G_{train}) \pm k\,\sigma(G_{train}) \tag{5}$$

The constant baseline is simpler than the time-dependent baseline, because it does not require to compute the trend line of the free energy, but is less sensitive to fluctuations. Figures 3 and 4 show a graphical representation of the time-dependent and constant baseline models, respectively, and the role of parameters σ and k. The green area in Figures 3-4 indicates the baseline of normal behavior computed from G_{train} with parameter $k = 3$.

D. Parameter Selector

The *parameter selector* validates both the time-dependent and constant baseline models against a reference dataset

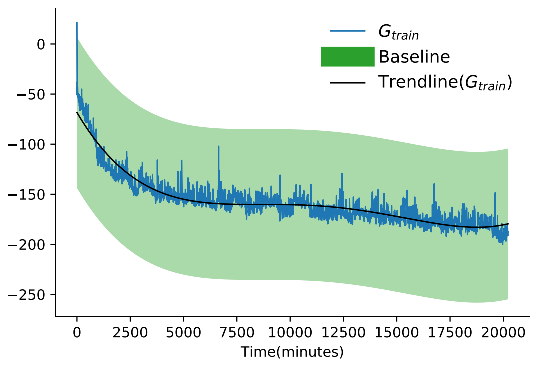

Fig. 3. Time-dependent baseline with trend line of degree 4 and $k = 3$

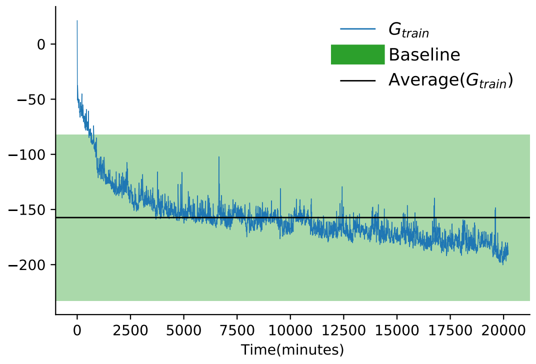

Fig. 4. Constant baseline with $k = 3$

with flagged anomalies, to select the most appropriate model for detecting failure-prone anomalies, as the model and the parameter k with highest precision and recall. We discuss the details of the validation of the two models and the selection of the best baseline model in Section III.

E. Anomaly Detector

The *anomaly detector* signals incoming failures as anomalies in the free energy, that is, deviations from the baseline model produced during training. The *anomaly detector* is based on the hypotheses that (i) erroneous states reflect in collective anomalies, as demonstrated by Mariani et al. [23], who provide evidence that erroneous behaviours produce errors that spread in the system, affecting a large set of KPIs, and (ii) the free energy represents the probability distribution that best fits the system behavior, thus the anomalous behaviors reflect in anomalous values of the free energy.

The *anomaly detector* computes the free energy $G(t)$ evaluating Equation (3) with the parameters optimised with the Parameter Selector. A value of $G(t)$ at time t that exceeds the baseline threshold indicates a suspect failure-prone state at time t. Figure 5 shows an example of free energy from an erroneous execution G_{faulty}, compared with the constant baseline computed from G_{train} in Figure 4: all values are above the baseline from Figure 4, thus they are marked in red as anomalous.

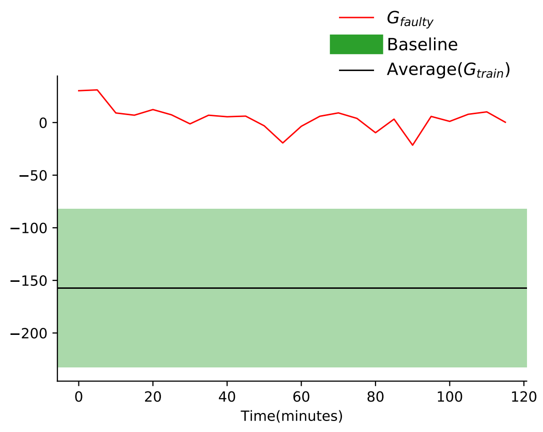

Fig. 5. Anomaly detection using the constant baseline with $k = 3$

III. Experimental Setting

In this section we introduce the research questions that we addressed to evaluate *EmBeD*, and the datasets that we used for selecting the baseline model, setting the parameters and answering the research questions. We discuss the experimental results in the next section.

A. Research Questions

We evaluated the effectiveness and performance of *EmBeD* by addressing the following research questions:

RQ1: *What is the precision of* EmBeD *in detecting failure-prone anomalies?*

RQ2: *What is the impact of* EmBeD *on a cloud environment?*

B. Datasets

We evaluated *EmBeD* with two datasets, *Yahoo Webscope* and *Clearwater*. We used the Yahoo Webscope dataset to produce the input reference dataset with normal behaviors for the *model builder* and the input reference dataset with flagged anomalies for the *parameter selector* in the training phase. We used the Clearwater dataset for validating the approach[1] .

Yahoo Webscope:

Yahoo Webscope dataset[2] is a reference library of datasets provided by Yahoo Research. We experimented with the data from folder *A1 Benchmark* in the S5 dataset of the Yahoo Webscope Program, which contains real production traffic data from various Yahoo! services with anomalies marked by experts, and thus represents a good reference dataset with flagged anomalies for the *parameter selector* to evaluate the baseline models and set parameter k. The A1 Benchmark dataset is composed of 67 files, one for each metric. Metrics are provided

[1]All data used in the experimental evaluation, including the original and post-processed Yahoo Webscope dataset and the Clearwater dataset are publically available at the link http://star.inf.usi.ch/star/software/embed/index.htm

[2]Yahoo Computing System Data. https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70. Last access: April 2018

as triples $\langle timestamp, metric\ value, label \rangle$, where a label 0 indicates a non-anomalous value, and a label 1 indicates an anomalous one. The files contain from 1420 to 1439 triples, that is, timestamps, and each timestamp corresponds to one hour of aggregated data. We considered the first 1420 triples in each file to have the same time range for all metrics, and combined the time series values from the different files in a single file for compatibility with the format of Table I.

We scaled the columns of the whole dataset to a normal distribution between 0 and 1 in two steps. We first rescaled each column of data with respect to its mean μ and standard deviation σ:

$$x_{standardized} = x - \frac{\mu(x)}{\sigma(x)} \tag{6}$$

according to the *feature standardization* procedure used in machine learning [32], and then we computed min-max scaling on each column to obtain the normalized input matrix in the format reported in Table I.

The Yahoo dataset is strongly unbalanced towards normal values: the values labeled as anomalous are less than 0.1% of the total. Unbalanced data usually negatively impact on the performance of the algorithm when used for classification and anomaly detection problems [33]: A good dataset for validating the baseline model should contain a statistically significant number of instances of anomalous values of the metrics that indicate failure-prone behavior.

Among the many techniques [33] that can be used to address the issue of unbalanced data, the most suitable techniques for obtaining good samples for validating the *EmBeD* model building phase are *undersampling* techniques. Undersampling consists in pre-processing the data by replicating the few anomalous instances, to produce a synthetic dataset similar to the initial one, but with more anomalous values [33]. We undersampled the data by randomly duplicating abnormal values labeled as anomalous, without introducing regular patterns.

The resulting dataset is a single matrix with 35 metrics, 1420 timestamps, and 244 anomalous values ($\sim$ 0.5% of the total). We divided the dataset in (i) a *training set* of 829 timestamps containing anomaly-free data, with less than 0.01% of anomalous values (ii) a *normal validation set* of 244 timestamps without anomalous values, and (iii) an *anomalous test set* of 244 timestamps, with only anomalous values, that is with at least 20% anomalous KPIs at each timestamp.

We used the *training set* to build the baseline model with the *model builder* and both the *normal validation* and *anomalous test sets* with the *parameter selector* to select the most accurate baseline model and parameters.

Clearwater Dataset:

With *Clearwater dataset*, we indicate a testbed that runs Clearwater[3], and that has been recently proposed by Sauvanaud et al. [2, 34] and Mariani et al. [23] to evaluate anomaly detectors. Clearwater is an open source implementation of the IP Multimedia Subsystem designed to be deployed in the cloud, and provides SIP-based call control for voice and video communications and for SIP-based messaging applications [2]. We experimented with a cluster managed with OpenStack, version Icehouse[4]. The cluser consists of 8 servers running Ubuntu 14.04 LTS that share a local subnet, composed of: (i) six *computing nodes* that run the KVM hypervisor[5] and operate the Clearwater Virtual Machine (VM) instances, (ii) a *controller node* that manages the VMs and provides other services such as identity services, image service and dashboard, and (iii) a *network node* that routes virtual networks through NAT. We generated the workload with the SIPp open source SIP traffic generator[6], by using a classic traffic pattern also used by Mariani et al. in their paper [23]: higher number of calls in weekdays than during weekends, growing amount of calls at daytime and decreasing at nighttime, with peaks at 9am and 7pm.

We collected 25 metrics from 14 resources, for a total of 350 KPIs, at operating system, machine, communication interfaces, and application levels. As an example, Table I reports some KPIs collected from the Clearwater virtual machines *bono*, *homestead* and *ellis*. We sampled the metric data every minute and aggregated them over time windows of 5 minutes.

We collected data both under normal execution conditions and with seeded faults, to simulate both normal executions and faulty executions that we used for validating the approach. We seeded faults of the three most common fault types in cloud systems: memory leak, packet loss and CPU hog [2, 35, 5, 36, 37, 38, 39] with Chaos Monkey[7] following three activation patterns, constant, random and exponential. To increase the generality of the results, we seeded faults in all core components of ClearWater, namely Bono, Sprout and Homestead nodes, and repeated the injection four times for each type of fault, producing a total of 108 experiments, organized in:

(i) a *training set* containing two weeks of normal executions, with 4045 timestamps,
(i) a *normal validation set* from 60 normal executions of 24 timestamps each, retrieved from the training set with cross-validation,
(i) an *anomalous test set* from 60 failure-prone executions, among which 31 with packet loss, 15 with CPU hog and 14 with memory leak. Each execution data covers a time of 120 minutes, aggregated in 24 timestamps.

We used the *training set* to build the baseline model with the *model builder* and both the *normal validation* and *anomalous test sets* to validate the effectiveness of the *anomaly detector* in predicting failures. We selected the baseline model and the parameters according to the data produced with the *Yahoo*

[3]Project Clearwater, http://www.projectclearwater.org. Last access: October 2018

[4]The OpenStack Project. https://www.openstack.org. Last access: October 2018

[5]The KVM Project. https://www.linux-kvm.org. Last access: October 2017.

[6]SIPp, http://sipp.sourceforge.net. Last access: October 2018.

[7]Chaos Monkey. https://github.com/Netflix/SimianArmy/wiki/Chaos-Monkey. Last access: October 2018.

Webscope dataset. Thus, we validated *EmBeD* on the Clearwater dataset by building the model with a dataset obtained by executing the system under normal conditions, without seeded faults, and by using the seeded faults only to validate the ability of *EmBeD* to detect failures.

We rescaled the Clearwater training and test set via standardization using Equation (6), and computed min-max scaling on the standardized dataset. The normalization on the normal validation set derives from splitting the training set for cross-validation. We used holdout instead of k-fold cross validation [40] because it is recommended for time series to preserve the temporal order of data and avoid bias in the classification. We identify a failure of Clearwater with a drop of success call rate below 60%, to exclude call rate drops dependent on the injection patterns.

IV. Results

We executed two sets of experiments taking advantage of the different nature of the two datasets. We used the Yahoo Webscope dataset to select both the baseline model and the parameter k, since the Yahoo dataset is labeled with anomalies from real systems, manually identified by Yahoo experts, and is thus well suited for tuning the model. We did not use the Yahoo dataset for validating the anomaly detector, due to both the lack of data about relations of anomalies with failures and the difficulty of splitting the dataset into independent subsets large enough to be statistically relevant.

We used the Clearwater dataset to validate *EmBeD*, since we have data from both normal executions and executions with seeded faults, the former useful to train the model, and both sets useful to validate the anomaly detector. We validated the anomaly detector on the Clearwater dataset with the parameters identified with the Yahoo dataset, thus validating the hypothesis that *EmBeD* can successfully detect failure-prone anomalies for a cloud system, without data about anomalies and failures in the training dataset.

Baseline Model and Parameter

We tuned both the choice of the baseline model and parameter k by experimenting with the Yahoo Webscope dataset, which contains flagged anomalies.

We evaluated both the *time-dependent* and *constant* baseline models that we computed according to Equations (4) and (5), respectively, for different values of the learning rate, $l = \{0.1, 0.01, 0.001\}$, and multiplicative factor, $k = \{1, 2, 3\}$. The learning rate is a standard parameter for tuning neural networks, and may strongly impact on both the model performance and the result. We evaluated the baseline models with different values of l, and did not observe any significant influence of l on the model, thus we considered the standard values $\{0.1, 0.01, 0.001\}$ for the learning rate l [30]. We considered the values $k = \{1, 2, 3\}$ since $k \in \mathbb{N}$ and higher values of k did not impact the result. We built all models by training the RBM with the training set, and selected the most suited model and parameter value with holdout cross-validation, by using the normal validation set and the anomalous test set.

TABLE II
FPR and TPR for the two baseline models and for different values of learning rate l and tolerance factor k

		Average $\pm k * \sigma$		Trend line $\pm k * \sigma$	
l	k	FPR	TPR	FPR	TPR
0.001	1	0.34	0.91	0.33	0.91
	2	0.07	0.83	0.07	0.84
	3	0.00	0.71	0.00	0.71
0.01	1	0.52	1.00	0.59	1.00
	2	0.20	1.00	0.34	1.00
	3	0.01	1.00	0.02	1.00
0.1	1	0.46	1.00	0.68	1.00
	2	0.03	1.00	0.25	1.00
	3	**0.00**	**1.00**	0.07	1.00

In all experiments we define the standard classification functions: (i) a false positive is an anomalous free energy value computed on the normal validation set, (ii) a true positive is an anomalous free energy value computed on the anomalous test set, (iii) a false negative is a non-anomalous free energy value computed on the anomalous test set, and (iv) a true negative is a non-anomalous free energy value computed on the normal validation set.

Table II shows the false positive rate (FPR) and true positive rate (TPR) for the two baseline models (*constant* and *time-dependent*), for different values of parameters l and k, highlighting the best values for FPR (the lowest) and TPR (the highest) in bold. The values indicate the constant baseline interval centered on the average free energy value (Equation (5)) with parameter values $l = 0.1$ and $k = 3$ as the most suited baseline model.

We evaluated the parameters corresponding to the best baseline model on the Yahoo data only, and we then used the identified parameters and model when experimenting with the Clearwater cluster, to both avoid overfitting and verify that the baseline model is independent on the target system. The results reported in the next subsections indicate that the model parameters $l = 0.1$, $k = 3$ lead to an f1 score greater than 0.94 on the Clearwater cluster, thus showing that the choice of the baseline model and the parameter k based on the Yahoo Webscope data is also a valid choice for a different system.

RQ1: Precision in detecting failure-prone anomalies

We evaluated the precision of *EmBeD* on the Clearwater dataset, by training with failure-free data and then validating with data from failure-free and failure-prone executions, the latter obtained with fault injection as described in Section III-B.

We trained *EmBeD* with data from the Clearwater training set, 14 days of normal executions sampled every 5 minutes, with a total of 4045 timestamps. We built the baseline model with the *model builder* with parameters $l = 0.1$ and $k = 3$, according to the results with the Yahoo dataset, thus without requiring information about anomalies. We then executed the *anomaly detector* on the 60 executions of 24 timestamps each of the normal validation set, to study true negatives and false positives, and on the 60 failure-prone executions of the anomalous test set, to study true positives and false negatives, according to the definition of classification functions used for the Yahoo dataset.

In line with results confirmed by many authors that indicate that failure-prone anomalies persist over time before the actual failures [22, 23, 41], we measure precision and recall by considering the persistency of revealed anomalies over time, for windows of 10, 50, 75 and 120 minutes. Since we sampled data every 5 minutes, the chosen time windows correspond respectively to 2, 10 and 15 consecutive samples of data with anomalous behavior. In all the experiments, the considered time windows are shorter than the time distance between the activations of the seeded errors and the system failures, thus failure alerts issued after even the largest considered time window would successfully predict the failure before its actual occurrence.

Table III shows the cumulative results of the experiments with cross-validation on both the normal validation set and the anomalous test set, for the considered time windows of 10, 50, 75 and 120 minutes.

TABLE III
CLEARWATER CLUSTER RESULTS OVER TIME (%)

time	TPR	FPR	FNR	TNR	precision	f1-score	accuracy
10'	100.00	0.90	0.00	99.10	90.23	94.86	99.17
50'	100.00	1.32	0.00	98.68	96.93	98.44	99.07
75'	100.00	1.60	0.00	98.40	97.51	98.74	99.02
120'	99.93	2.15	0.07	97.85	97.89	98.90	98.89

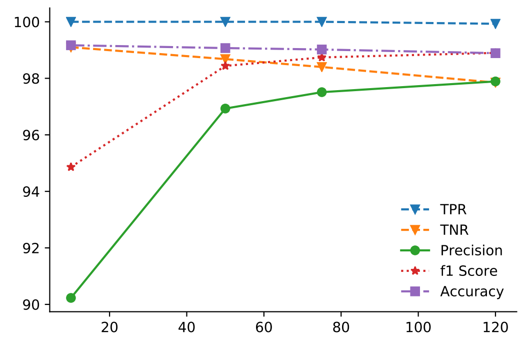

Fig. 6. Variability of TPR, TNR, precision, f1 score and accuracy over time

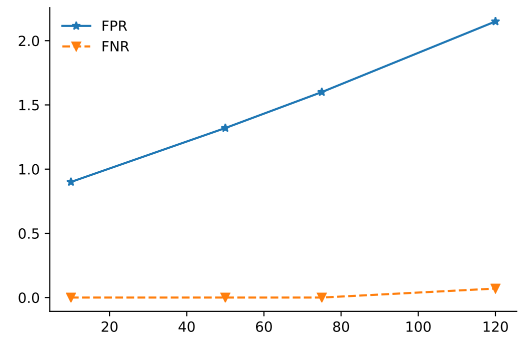

Fig. 7. Variability of FPR and FNR over time

Figures 6 and 7 plot the results reported in Table III over time to better visualize the trend of true positive rate, true negative rate, f1 score, precision, accuracy (Figure 6), false positive rate and false negative rate (Figure 7).

The results confirm that *EmBeD* can effectively predict failures by early revealing failure-prone anomalies. The very high true positive rate (TPR) that we compute for the anomalous dataset confirms that *EmBeD* precisely predicts all failures already after 2 samples and the prediction remains stable for many samples. The slight drop over time in accuracy and true negative rate (TNR) indicates that the noise in the data in the proximity of the failure may slightly affect the prediction, if the alerts are not followed by proper corrective actions. The very low false positive rate (FPR) that we compute for the anomalous dataset indicates that *EmBeD* rarely misses to predict failures before their occurrence. The slight increase over time, from a negligible 0.9% after 2 samples to 2.15% after 24 consecutive samples indicates that in very few cases, a slightly anomalous frequency of collective anomalies over a long period may mistake the anomaly detector, and the effect may slightly inflate in the presence of some particular combinations of collective anomalies over a long period. The very good values of false negative rate (FNR almost null) and true negative rate (TNR over 97%), which we compute for the normal validation dataset, indicate the excellent ability of distinguishing normal behaviors with only a negligible amount of false alarms. The small variations over long time windows are due to the impact of noisy data over long time series, as for TPR and FPR values.

Both precision and f1-score noticeably increase from 2 to 10 samples (from 10 to 50 minutes time windows) and remain almost stable afterwards, confirming excellent ability of timely predicting failures. By averaging over the different time windows, the experimental results indicate that *EmBeD* detects anomalous behaviors that may lead to failures with 95.64% precision, 99.98% recall (TPR), 99.04% accuracy and a false positive rate of 1.49%. *EmBeD* distinguished failure-prone from correct behaviour with 31 false negatives among all timestamps in the failure-free data, and one false positive in one experiment with memory leak. False negatives are values of the free energy that exceed the baseline interval without corresponding to failure-prone behaviors. In our experiments, the few false negatives derive from fluctuations in the free energy that slightly exceed the baseline model, and may be related to the fact that the parameters of the baseline model are tuned on another dataset. The good values of false positive rate and true positive rate indicates that *EmBeD* can detect anomalous patterns related to unknown faults.

RQ2: Impact of EmBeD *on the cloud environment*

The impact of *EmBeD* on the cloud environment depends on (i) the time required for training the RBM, updating the model and computing the free energy, and (ii) the runtime overhead. The former time impacts on the ability of detecting failure-prone anomalies early enough to activate preventing actions, while the latter affects the overall performance of the cloud system.

We minimized the time required for training the RBM, updating the model and computing the free energy by following Hinton's recommendations [30] of optimizing the computational time by training the RBM with Persistent Contrastive Divergence, and computing the free energy through a formula linear in the number of hidden units. We evaluated *EmBeD* with a number of hidden units equal to the size of the input vector, thus equal to the number of metrics collected from the system (35 for the Yahoo dataset, 350 for the Clearwater cluster). We implemented *EmBeD* on GPUs to reduce the time for training and testing. When executed on a dedicated 16 GB RAM laptop with 3840 NVIDIA CUDA cores, *EmBeD* trained the RBM and computed the free energy in average 4 seconds on the Yahoo dataset, and 24 seconds on Clearwater dataset. When deployed in a cloud infrastructure, *EmBeD* can be executed on a machine independent from the cloud system, thus limiting the runtime overhead to the monitoring probes that collect data from the production environment. According to the experiments of Mariani et al. [23], the overhead of the monitoring resources amounts to an average increase of 2.63% on CPU usage and of 1.91% on memory usage, which is negligible for cloud services.

Threats to Validity

The main threats to the external validity of *EmBeD* derive from the choice of only one dataset (the Clearwater dataset) for validating *EmBeD*, the sensitivity of the model computed with the RBM for training the parameters, and the normalization strategy of the input data. The main threats to the internal validity depend on the the datasets chosen for validation, and the generalizability and scalability of the results.

Experimental dataset:

We carried on the experiments on two datasets. We chose the Yahoo A1 Benchmark and the Clearwater datasets as two realistic examples, because they both contain real values from networked metric data. We used the Yahoo dataset, proposed as a benchmark for the research community, for tuning the parameters σ and k, and the Clearwater dataset for validation. Even though we describe the Clearwater dataset as a single dataset, we produced a varied and statistically significant dataset by testing with three types of faults, experimenting with three different fault seeding mechanisms, and collecting data from 60 different runs. Clearwater has been used in related work [34] and thus can be considered as a reference benchmark for testing anomaly detection techniques in cloud systems.

Sensitivity of the model for parameter training:

The results of training a RBM depend on the choice of the numerical meta-parameters such as the learning rate, the momentum, the weight-cost, the sparsity target, the initial values of the weights, the number of hidden units and the size of each mini-batch. We chose the best parameters to avoid overfitting and errors following Hinton's rules [30].

We chose the size of mini-batches inside the interval $[10, 100]$ to minimize to the sampling error, and we avoided overfitting of the model by verifying that the free energy of the training data was comparable to the free energy of (non-anomalous) validation data.

Scalability and generalizability:

We evaluate *EmBeD* on a cluster of eight nodes, which is in range with the experimental setting reported in related literature, in particular, it is smaller than the 20 nodes cluster used in Gulenko et al.'s experiments [42], and four times larger than the cluster used in Sauvanaud et al.'s experiments [34]. We use the same Clearwater configuration as Sauvanaud et al., since the configuration details in Sauvanaud et al.'s paper allow for replicating the configuration, while other papers do not provide enough details.

We address the scalability of *EmBeD* by considering two datasets that differ in size by an order of magnitude. The flexible architecture of the RBM in *EmBeD* allows to adjust the size of the hidden layer to maximize the performance also with big datasets.

V. Related work

Ibidunmoye et al.'s survey paper [7] provides an excellent overview of approaches to detect anomalies in cloud systems.

Anomaly detectors that exploit *statistical analysis* are usually based on parametric goodness-of-fit tests like generalized Student's t-test [43], chi squared test [44], and non-parametric tests such as the Kolmogorov-Smirnov test [45], or data density, probability density [11], and relative entropy [46, 47]. Statistical analysis does not require any prior knowledge on the system behavior, nor extensive training sessions to derive models, and well identifies the many anomalous combinations of metric values. However, hypothesis testing can give biased results if the underlying hypothesis on the data is not accurate, leading to many false positives, thus it is not suitable for distinguishing benign anomalies, originated from data fluctuation, from failure-prone anomalies that are related to faults.

Machine learning techniques infer models of normal and/or faulty behavior, and identify failure-prone anomalies from the inferred models by exploiting supervised, unsupervised, semi-supervised, or deep learning (either supervised or unsupervised) [7]. Approaches that exploit supervised learning require labeled instances for both normal and anomalous behavior, and thus require knowledge of faulty signatures, which is usually obtained by training models with seeded faults [34, 13, 14, 15, 16, 17]. These approaches can be extremely precise, but seeding faults in production environment dramatically interferes with the system behavior, and is rarely feasible in production systems. Supervised approaches are sensitive to the learned models, and fail to detect failure-prone behaviors not previously encoded in the models.

Approaches that exploit unsupervised learning infer structures and signatures encoded in the unlabeled data used for training, and do not require fault seeding, but result in many false alarms, that is anomalies not related to failures. Increasing the precision and accuracy with intensive and long lasting training may incur in severe overhead.

Both Ahmed et al. [10] and Lakhina et al. [9] investigate Principal Component Analysis for analyzing network traffic data. Ibidunmoye et al. developed two techniques, *PAD* [11] and *BAD* [8], for combining statistical analysis and kernel density estimation (KDE) to highly unbalanced data, splitting the high cost of KDE estimation into small sliding windows. The accuracy of the techniques is sensitive to the optimal window size, which is empirically defined for each dataset. *SORAD* [48] implements a simple regression algorithm for detecting some kinds of anomalies, similarly to the Hierarchical Temporal Memory (HTM) algorithm [49] and *ADVec*, the ARIMA-based detector adopted at Twitter [50], albeit with a smaller detection time. ClouDiag [12], Surus [8], UBL [35], PREPARE [5], EaAD [51] and Guan et al. [3] exploit unsupervised or semi-supervised machine learning for cloud reliability.

Semi-supervised techniques [35, 51] require minimal labelling of the input data, they can train models with normal instances only, and signal anomalies when they detect behaviors that deviate from the inferred model. They may detect new types of faults without requiring fault seeding, thus overcoming the main limitations of supervised learning. *UBL* [35] exploits Self-Organizing Maps (SOMs), a particular type of Artificial Neural Network, to capture emergent behaviors and unknown anomalies with low false positive rate. SOMs require long training time, and the prediction is accurate only if failures manifest gradually and slowly. *PREPARE* [5] predicts anomalies in virtualized systems by combining a 2-dependent Markov model with a tree-augmented Bayesian network, which limits the applicability of PREPARE to specific types of faults. EeAD [51] exploits one-class, multi-class, imbalanced and online SVM, incorporating semi-supervised learning, to detect anomalous virtual machines in a cloud prototype. Guan et al. [3] propose an ensemble of Bayesian Network models to characterize the normal execution states of a cloud system, and to signal incoming failures when detecting states not encoded in the models.

The preliminary work by Gulenko et al. [42] implements an event-base technique for detecting anomalies at real-time. Sauvanaud et al. [2, 34] exploit classifiers such as Random Forest, Naive Bayes, Neural Network and Nearest Neighbors to detect and predict anomalies at metric level through Service Level Agreement (SLA) violations. Sauvanaud et al.'s techniques detects anomalies at metric level, a much finer granularity level than *EmBeD*.

[8]Netflix Surus, https://github.com/Netflix/Surus Last access: April 2018

Unsupervised and semi-supervised techniques can effectively reduce the low false positive rate only with high computational cost, especially in the presence of large volumes of data [35, 12, 5].

Deep architectures address the limitations of unsupervised anomaly detection, and are far more efficient than standard unsupervised algorithms when dealing with big data. However, to authors' knowledge there is no application of RBMs or deep learning to proactive detection of failure-prone anomalies in cloud systems. Recently, Schneider et al. applied RBMs to fault detection and root cause analysis in virtualized environments [52]. Liu et al. [24] exploit the RBM and the free energy to classify outliers in complex interacting systems. Do et al. [53] show that an energy-based model computed with a RBM detects efficiently anomalies from mixed-type data. *EmBeD* differs from Do et al.'s approach in terms of main application, algorithmic structure, technical evaluation and overall performance: (i) Do et al. apply RBM to detect outliers as mere deviations from the norm, *EmBeD* relates out-of-norm values of the free energy to failure-prone behaviors in cloud systems, (ii) the algorithmic architecture of *EmBeD* is simpler than the one of Do et al. since *EmBeD* exploits a single RBM, while Do et al. exploit a stack of RBMs, (iii) we designed *EmBeD* for detecting failures online and we validate it as such, while Do et al. validate their approach only on batch datasets, (iv) *EmBeD* improves the f1-score from the 0.91 score of Do et al.'s approach to an f1-score between 0.95 and 0.99, as reported in Table III.

EmBeD combines the advantages of anomaly detection techniques based on a single distribution over the input data, and the advantages of deep learning by leveraging the RBM as a very simple building block of deep architectures, which can be trained and updated at runtime with negligible overhead in production environment.

VI. Conclusion

In this paper, we propose *EmBeD*, an energy-based anomaly detection approach that exploits the free energy computed with a Restricted Boltzmann Machine on KPIs to detect failure-prone anomalies in cloud systems. *EmBeD* does not require training with seeded faults, and the experimental results reported in the paper confirm that *EmBeD* can accurately and timely predict failures, with a negligible amount of false positives and very little overhead on the system at runtime. Thus *EmBeD* overcomes the main limitations of current statistical and machine learning based approaches, with excellent results, which are in line if not better than similar preliminary work that studies alternative deep learning approaches in different environmental settings.

The main contribution of this work is the proposal of RBM as a viable technique to precisely and timely identify collective KPI anomalies strongly related to faulty states that can lead to failures if not properly fixed, and a set of experiments on representative benchmarks that confirm the high precision and recall of the approach.

References

[1] X. Chen, C.-D. Lu, and K. Pattabiraman. “Failure Analysis of Jobs in Compute Clouds: A Google Cluster Case Study”. In: *Proc. International Symposium on Software Reliability Engineering*. IEEE, 2014, pp. 167–177.

[2] C. Sauvanaud, K. Lazri, M. Kaâniche, and K. Kanoun. “Anomaly Detection and Root Cause Localization in Virtual Network Functions”. In: *Proc. International Symposium on Software Reliability Engineering*. IEEE, 2016, pp. 196–206.

[3] G. Qiang, Z. Ziming, and F. Song. “Ensemble of Bayesian Predictors for Autonomic Failure Management in Cloud Computing”. In: *Proc. International Conference on Computer Communications and Networks*. IEEE, 2010, pp. 1–6.

[4] A. W. Williams, S. M. Pertet, and P. Narasimhan. “Tiresias: Black-box failure prediction in distributed systems”. In: *Proc. International Parallel and Distributed Processing Symposium*. IEEE, 2007, pp. 1–8.

[5] T. Yongmin, N. Hiep, S. Zhiming, G. Xiaohui, V. Chitra, and R. Deepak. “PREPARE: Predictive Performance Anomaly Prevention for Virtualized Cloud Systems”. In: *Proc. International Conference on Distributed Computing Systems*. IEEE, 2012, pp. 285–294.

[6] S. Jin, Z. Zhang, K. Chakrabarty, and X. Gu. “Changepoint-based anomaly detection in a core router system”. In: *Proc. International Test Conference*. IEEE, 2017, pp. 1–10.

[7] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth. “Performance Anomaly Detection and Bottleneck Identification”. In: *Computing Surveys* 48.1 (2015), 4:1–4:35.

[8] O. Ibidunmoye, A.-R. Rezaie, and E. Elmroth. “Adaptive Anomaly Detection in Performance Metric Streams”. In: *Transactions on Network and Service Management* 15.1 (2018), pp. 217–231.

[9] A. Lakhina, M. Crovella, and C. Diot. “Diagnosing Network-wide Traffic Anomalies”. In: *Proc. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. New York, NY, USA: ACM, 2004, pp. 219–230.

[10] T. Ahmed, M. Coates, and A. Lakhina. “Multivariate Online Anomaly Detection Using Kernel Recursive Least Squares”. In: *Proc. International Conference on Computer Communications*. IEEE, 2007, pp. 625–633.

[11] O. Ibidunmoye, T. Metsch, and E. Elmroth. “Real-time detection of performance anomalies for cloud services”. In: *Proc. International Symposium on Quality of Service*. IEEE/ACM, 2016, pp. 1–2.

[12] H. Mi, H. Wang, Y. Zhou, M. Lyu, and H. Cai. “Toward Fine-Grained, Unsupervised, Scalable Performance Diagnosis for Production Cloud Computing Systems”. In: *Transactions on Parallel and Distributed Systems* 24.6 (2013), pp. 1245–1255.

[13] D. Novaković, N. Vasić, S. Novaković, D. Kostić, and R. Bianchini. “DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments”. In: *USENIX Annual Technical Conference*. ACM, 2013, pp. 219–230.

[14] X. Pan, J. Tan, S. Kavulya, R. Gandhi, P. Narasimhan, and T. Jiaqi. “Ganesha: blackBox diagnosis of MapReduce systems”. In: *ACM SIGMETRICS Performance Evaluation Review* 37.3 (2010), pp. 8–13.

[15] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. “Pinpoint: Problem determination in large, dynamic internet services”. In: *Proc. International Conference on Dependable Systems and Networks*. IEEE, 2002, pp. 595–604.

[16] C. Yuan, N. Lao, J.-R. Wen, J. Li, Z. Zhang, Y.-M. Wang, and W.-Y. Ma. “Automated Known Problem Diagnosis with Event Traces”. In: *Proc. SIGOPS EuroSys European Conference on Computer Systems*. ACM, 2006, pp. 375–388.

[17] S. Zhang, I. Cohen, M. Goldszmidt, J. Symons, and A. Fox. “Ensembles of models for automated diagnosis of system performance problems”. In: *Proc. International Conference on Dependable Systems and Networks*. IEEE, 2005, pp. 644–653.

[18] C. Monni and M. Pezzè. “Energy-Based Anomaly Detection: A New Perspective for Predicting Software Failures”. In: *Proc. International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track*. IEEE, 2019, to appear.

[19] D. Chandler. *Introduction to Modern Statistical Mechanics*. Oxford University Press, Sept. 1987, p. 288.

[20] A. Pelissetto and E. Vicari. “Critical phenomena and renormalization-group theory”. In: *Physics Reports* 368.6 (2002), pp. 549–727.

[21] S. N. Dorogovtsev, A. V. Goltsev, and J. F. F. Mendes. “Critical phenomena in complex networks”. In: *Reviews of Modern Physics* 80 (2008), pp. 1275–1335.

[22] S. Jin, Z. Zhang, K. Chakrabarty, and X. Gu. “Accurate anomaly detection using correlation-based time-series analysis in a core router system”. In: *Proc. International Test Conference*. IEEE, 2016, pp. 1–10.

[23] L. Mariani, C. Monni, M. Pezzè, O. Riganelli, and R. Xin. “Localizing Faults in Cloud Systems”. In: *Proc. international Conference on Software Testing*. IEEE, 2018, pp. 262–273.

[24] C. Liu, S. Ghosal, Z. Jiang, and S. Sarkar. “An unsupervised anomaly detection approach using energy-based spatiotemporal graphical modeling”. In: *Cyber-Physical Systems* 3.1-4 (2017), pp. 66–102.

[25] N. L. Roux, N. Heess, J. Shotton, and J. Winn. “Learning a Generative Model of Images by Factoring Appearance and Shape”. In: *Neural Computation* 23.3 (2011), pp. 593–650.

[26] J. Kivinen and C. Williams. “Multiple Texture Boltzmann Machines”. In: *Proc. International Conference*

on Artificial Intelligence and Statistics. Vol. 22. PMLR, 2012, pp. 638–646.

[27] A. Fischer and C. Igel. "An Introduction to Restricted Boltzmann Machines". In: *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*. Springer, 2012, pp. 14–36.

[28] G. Montufar and N. Ay. "Refinements of Universal Approximation Results for Deep Belief Networks and Restricted Boltzmann Machines". In: *Neural Computation* 23.5 (May 2011), pp. 1306–1319.

[29] N. L. Roux and Y. Bengio. "Representational Power of Restricted Boltzmann Machines and Deep Belief Networks". In: *Neural Computation* 20.6 (2008), pp. 1631–1649.

[30] G. E. Hinton. "A Practical Guide to Training Restricted Boltzmann Machines". In: *Neural Networks: Tricks of the Trade: Second Edition*. Springer, 2012, pp. 599–619.

[31] A. Fischer and C. Igel. "Training Restricted Boltzmann Machines". In: *Pattern Recognition* 47.1 (2014), pp. 25–39.

[32] J. Grus. *Data Science from Scratch: First Principles with Python*. O'Reilly Media, Inc., 2015.

[33] P. Branco, L. Torgo, and R. P. RIbeiro. "A Survey of Predictive Modelling on Imbalanced Domains". In: *ACM Computing Surveys* 49.2 (2016), 31:1–21:50.

[34] C. Sauvanaud, M. Kaâniche, K. Kanoun, K. Lazri, and G. Silvestre. "Anomaly detection and diagnosis for cloud services: Practical experiments and lessons learned". In: *Journal of Systems and Software* 139 (2018), pp. 84–106.

[35] D. J. Dean, H. Nguyen, and X. Gu. "UBL: Unsupervised Behavior Learning for Predicting Performance Anomalies in Virtualized Cloud Systems". In: *Proc. International Conference on Autonomic Computing*. ACM, 2012, pp. 191–200.

[36] J. Tan, X. Pan, E. Marinelli, S. Kavulya, R. Gandhi, and P. Narasimhan. "Kahuna: Problem diagnosis for mapreduce-based cloud computing environments". In: *Proc. Conference on Network Operations and Management Symposium*. IEEE, 2010, pp. 112–119.

[37] G. Wang and T. S. E. Ng. "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center". In: *Proc. International Conference on Computer Communications*. IEEE, 2010, pp. 1–9.

[38] Y. Dai, Y. Xiang, and G. Zhang. "Self-healing and Hybrid Diagnosis in Cloud Computing". In: *Proc. International Conference on Cloud Computing Technology and Science*. IEEE, 2009, pp. 45–56.

[39] B. Sharma, P. Jayachandran, A. Verma, and C. R. Das. "CloudPD: Problem determination and diagnosis in shared dynamic clouds". In: *Proc. International Conference on Dependable Systems and Networks*. IEEE, 2013, pp. 1–12.

[40] R. Kohavi. "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection". In: *Proc. International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 1995, pp. 1137–1143.

[41] L. Mariani, M. Pezzè, and M. Santoro. "GK-Tail+ An Efficient Approach to Learn Software Models". In: *Transactions on Software Engineering* 43.8 (2017), pp. 715–738.

[42] A. Gulenko, M. Wallschläger, F. Schmidt, O. Kao, and F. Liu. "A System Architecture for Real-time Anomaly Detection in Large-scale NFV Systems". In: *Procedia Computer Science* 94 (2016), pp. 491–496.

[43] J. Hochenbaum, O. S. Vallis, and A. Kejariwal. "Automatic Anomaly Detection in the Cloud Via Statistical Learning". In: *ArXiv e-prints* (2017).

[44] M. Solaimani, M. Iftekhar, L. Khan, and B. Thuraisingham. "Statistical technique for online anomaly detection using Spark over heterogeneous data from multi-source VMware performance data". In: *Proc. International Conference on Big Data*. IEEE, 2014, pp. 1086–1094.

[45] G. S. Smrithy and R. Balakrishnan. "A Statistical Technique for Online Anomaly Detection for Big Data Streams in Cloud Collaborative Environment". In: *Proc. International Conference on Computer and Information Technology*. IEEE, 2016, pp. 108–111.

[46] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan. "Statistical techniques for online anomaly detection in data centers". In: *Proc. International Symposium on Integrated Network Management (IM 2011) and Workshops*. IFIP/IEEE, 2011, pp. 385–392.

[47] S. Roy, A. C. König, I. Dvorkin, and M. Kumar. "PerfAugur: Robust diagnostics for performance anomalies in cloud services". In: *Proc. International Conference on Data Engineering*. IEEE, 2015, pp. 1167–1178.

[48] M. Thill, W. Konen, and T. Bäck. "Online anomaly detection on the webscope S5 dataset: A comparative study". In: *Evolving and Adaptive Intelligent Systems*. IEEE, 2017, pp. 1–8.

[49] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha. "Unsupervised real-time anomaly detection for streaming data". In: *Neurocomputing* 262 (2017), pp. 134–147.

[50] J. Hochenbaum, O. S. Vallis, and A. Kejariwal. "Automatic Anomaly Detection in the Cloud Via Statistical Learning". In: *CoRR* abs/1704.07706 (2017).

[51] G. Wang and J. Wang. "An Anomaly Detection Framework for Detecting Anomalous Virtual Machines under Cloud Computing Environment". In: *International Journal of Security and Its Applications* 10.1 (2016), pp. 75–86.

[52] C. Schneider, A. D. Barker, and S. A. Dobson. "Autonomous fault detection in self-healing systems using Restricted Boltzmann Machines". In: *Proc. International Conference and Workshops on Engineering of Autonomic and Autonomous Systems*. IEEE, 2014.

[53] K. Do, T. Tran, and S. Venkatesh. "Energy-based anomaly detection for mixed data". In: *Knowledge and Information Systems* 57.2 (2018), pp. 413–435.