Anomaly Detection in Cloud-Native Systems

Francesco Lomio,[1] Sergio Moreschini,[1] Xiaozhou Li,[1] Valentina Lenarduzzi[2]
[1]Tampere University — [2]University of Oulu
francesco.lomio@tuni.fi; sergio.moreschini@tuni.fi; xiaozhou.li@tuni.fi; valentina.lenarduzzi@oulu.fi

***Abstract*—Companies develop cloud-native systems deployed on public and private clouds. Since private clouds have limited resources, the systems should run efficiently by keeping performance related anomalies under control. The goal of this work is to understand whether a set of five performance-related KPIs depends on the metrics collected at runtime by Kafka, Zookeeper, and other tools (168 different metrics). We considered four weeks worth of runtime data collected from a system running in production. We trained eight Machine Learning algorithms on three weeks worth of data and tested them on one week's worth of data to compare their prediction accuracy and their training and testing time. It is possible to detect performance-related anomalies with a very high level of accuracy (higher than 95% AUC) and with very limited training time (between 8 and 17 minutes).Machine Learning algorithms can help to identify runtime anomalies and to detect them efficiently. Future work will include the identification of a proactive approach to recognize the root cause of the anomalies and to prevent them as early as possible.**

***Index Terms*—Anomaly Detection, Kafka metrics, Machine Learning, Empirical Study**

I. Introduction

In the past years it has become more common for companies to develop large-scale projects using microservices. This decision is usually driven by their benefits, including increased maintainability of the systems, development and deployment independence between teams, and many other reasons. However, microservice–based systems have many more moving parts compare to monolithic one. In such a complex system, runtime failures are unavoidable [1] and must be kept under control.

Such systems are commonly deployed on public and private clouds. As private clouds often have limited resources, the systems should always maintain an optimal level of performance, and therefore their load and their status needs to be constantly monitored. For this specific work, we used Kafka, Zookeeper, Prometheus and other tools to actively monitoring all the services composing the systems. The monitoring tools collect 168 different metrics, including performance-related metrics, hardware failures, and metrics related to the communication between services, such as throughput and message lags.

Our goal is to understand whether it is possible to predict anomalies from the different services composing the system that can degrade their performance, so as to take actions before it decreases significantly or before the system fails. An anomaly can be defined as a rare event that occur in an otherwise normal data. This event usually differs significantly from the rest of the data. An example could be an anomalous usage of the memory resources by some of the components of the system [2]

In our case, the hardware usage has to be optimized for our system. The services, in fact, are usually running very closely to the maximum hardware capacity, especially when running on private cloud systems. A few of the performance-related metrics, are usually reported in Prometheus with a delay of two to five seconds. Predicting their values based on the other available metrics would enable to spot possible issues early enough, so that healing mechanisms could be activated.

Different anomaly detection techniques have been proposed in the literature. Data-driven techniques are based on the analysis of data collected at runtime and are designed to predict anomalies in complex systems based on abnormal behaviour of a system [3]. Researchers have proposed both supervised and unsupervised Machine Learning techniques for anomaly detection. Unsupervised models are trained based on data from the correct execution of a system; however, they are less accurate than supervised ones. Supervised models train the model under consideration of both normal and failing execution data [4] [5].

Besides accuracy, the time required by Machine Learning algorithms to train must also be addressed [6]. Our system produces a huge amount of data every day, and training the system on this data could result in very high costs in terms of time and resources, requiring to reduce the amount of data collected to the most informative one [7].

In order to understand which Machine Learning technique might be most suitable for our purpose, we compared eight different Machine Learning algorithms that can be applied to our runtime data by training them continuously (e.g., every week) to correctly predict anomalies with the lowest training cost (in terms of time) possible. This work will contribute to the body of knowledge of industrial experience on anomaly detection, helping companies working with cloud-native systems based on similar technologies as well as researchers to understand how the different techniques perform and to conduct empirical studies in industry.

The remainder of this paper is structured as follows. In Section II, we describe the basic concepts underlying this work. In Section III, we report the design of our empirical study. In Section IV, we present the achieved results and we discuss them in Section V. In Section VI we highlight the threats to validity. Finally, in Section VII we presents some related work done by researchers in recent years and in Section VIII, we discuss the results and draw conclusions.

II. Background

A. Microservices

Microservices are small and autonomous services deployed independently, with a single and clearly defined purpose [8], [9]. In microservices-based systems, each service can be developed using different languages and frameworks. Each service is deployed to their dedicated environment whatever efficient for them.

The communication between the services can be based on either REST or message queue. So, whenever there is a change in business logic in any of the services, the others are not affected as long as the communication endpoint is not changed. As a result if any of the components of the system fails, the failure will not affect the other components or services, which is a big drawback of monolithic system [8].

B. Anomaly detection and Machine Learning techniques

Anomaly detection [10] addresses the problem of finding patterns in data with unexpected behavior, called anomalies or outliers. Anomaly detection is applied to multiple domains such as fraud detection, intrusion detection, and health care. Because each domain has different data and different approaches, during the years many different techniques have been developed and they can be summarized into six categories: classification-based, clustering-based, nearest neighbor-based, statistical techniques, information theoretic techniques, and spectral techniques.

Because of the nature of data, classification-based techniques have been used in this study. The main advantages of the Classification-based techniques are a fast testing phase and the availability of powerful algorithms for classification. The main disadvantages are the need for accurate labels and that the output is only a label and therefore it is not possible to have a score.

More specifically, we compared eight machine learning classifiers, which are: Logistic Regression [11], a Decision Tree [12], and 6 *ensemble classifiers*, Bagging [13], Random Forest [14], Extremely Randomized Trees [15], AdaBoost [16], Gradient Boosting [17] and XGBoost [18], an optimized implementation of Gradient Boosting.

III. Empirical Study Design

In this section, we will describe the case study design including the goal and the research questions, the study context, the data collection, and the data analysis procedure.

A. Goal and Research Questions

The goal of this work is to analyze eight machine learning techniques with respect to their performance-related anomaly detection accuracy, training and testing time, in the context of clout-native systems. For this purpose, the following research questions were derived:

RQ1. Is there a Machine Learning algorithm that can accurately detect performance-related anomalies in cloud-native systems?

- RQ1.1. Which Machine Learning algorithm has higher accuracy in detecting performance-related anomalies in cloud-native systems?

RQ2. Which Machine Learning algorithm can accurately detect performance-related anomalies with the shortest training time?

RQ3. What are the most important metrics to be considered when detecting performance-related anomalies?

RQ4. How many components are necessary to accurately detect performance-related anomalies?

Metrics. We are continuously monitoring our system at runtime, collecting a total of 182 metrics from the following tools:

- Apache Kafka: 120 metrics referring to Kafka brokers, producers, and consumers
- Apache ZooKeeper: 20 metrics regarding its nodes
- Java Virtual Machine: 16 metrics for monitoring threads, classes, and memory of JVMs
- Process: 6 standard metrics for monitoring processes.
- Java Management Extension: 4 metrics regarding JMX configurations

151 out of the 168 collected metrics, are reported in Prometheus with a maximum delay of 200 milliseconds, while the remaining ones are reported with a delay that ranges between 500 milliseconds and 5 seconds.

In order to answer our Q1 and Q2, we first needed to identify a set of metrics that are symptoms of performance anomalies. For this purpose, we identified five KPIs that we consider fundamental in our system and whose thresholds should never be exceeded (see Table I. The five KPIs are reported in Prometheus with a delay that ranges between two and five seconds, corroborating the need to identify a prediction model to detect possible performance-related metrics earlier than when the metrics are shown in the system. The thresholds used are the one proposed by Prometheus

TABLE I
Dependent Variables (KPIs)

Metric	Description	Threshold
Min Fetch Rate	The minimum rate at which the consumer sends fetch requests to the broker. If a consumer is dead, this value drops to roughly 0.	> 0.5
% Network Processor Idling Time	Average fraction of time the network processor threads are idle. The values are between 0 (all resources are used) and 1 (all resources are available).	> 0.3
Max Message Lag	Maximum lag in messages between the follower and leader replicas.	< 50
Avg Request Latency	Amount of time it takes for the server to respond to a client request (since the server was started).	< 100
Request Queue Size	Number of requests queued in the server. Goes up when the server receives more requests than it can process.	< 10

B. Accuracy

To assess the detection accuracy of the different Machine Learning algorithms, we performed a 10-fold cross-validation, dividing the data into ten parts; *i.e.,* we trained the models ten times, always using 1/10 of the data as a testing fold. The data related to each project was split into ten sequential parts, thus respecting the temporal order and the proportion of data for each project. The models were trained iteratively on groups of data preceding the test set. For example, in fold 1, we used group 1 for training and group 2 for testing; in fold 2, groups 1 and 2 were used for training and group 3 for testing, and so on for the remaining folds.

As accuracy metrics, we first calculated precision and recall. However, as suggested by [19], these two measures present some biases as they are mainly focused on positive examples and predictions and do not capture any information about the rates and kinds of errors made.

The contingency matrix (also called confusion matrix) and the related f-measure help to overcome this issue. Moreover, as recommended by Powers [19], the Matthews Correlation Coefficient (MCC) should also be considered to understand any potential disagreement between the actual values and the predictions, as it involves all four quadrants of the contingency matrix.

From the contingency matrix, we retrieved the *true negative rate* (TNR) measure, which measures the percentage of negative samples correctly categorized as negative; the *false positive rate* (FPR), which measures the percentage of negative samples misclassified as positive; and the *false negative rate* (FNR), which measures the percentage of positive samples misclassified as negative. The *true positive rate* measure was left out as it is equivalent to the recall.

Finally, we calculated the Receiver Operating Characteristics (ROC) and the related Area Under the Receiver Operating Characteristic Curve (AUC), i.e., the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one. The choice of using the AUC was made in order to rank the machine learning models. As shown in [20] AUC can be used fairly accurately is the best accuracy measure that can be used to compare machine learning models. The ROC curve has been calculated and plotted thanks to the "roc_curve()" function of the SciKit-Learn [21] library.

C. Training and Testing Time

Regarding this aspect, we collected and compared the training and testing time (in seconds) for each algorithm. The goal is to be able to select one algorithm that can be trained with the shortest training and with a high level of accuracy.

D. Context

Our system is composed of several microservices running on top of Kubernetes and communicating using a lightweight message bus (Apache Kafka[1]). The size of our system requires multiple Kafka brokers and therefore the use of Zookeeper to coordinate the different Kafka instances. We collect different metrics from Kafka, Zookeeper, and from other tools we installed to monitor our system.

[1]Apache Kafka https://kafka.apache.org. Accessed: June 2019.

E. Data Collection and Preparation

The monitoring systems collect data every 60 seconds and store it in Prometheus. We downloaded the data for four weeks, querying Prometheus weekly from its APIs.

The result is a csv file reporting the different measures collected from the different instances of the services with their related time stamps. An example of the data structure is reported in Table II.

TABLE II
THE DATA COLLECTED WEEKLY

	Variables (168)								
	Kafka			**ZooKeeper**			**Others**		
Time stamp	M K1	...	M K120	M Z1	...	M Z22	M O1	...	M O26
T_1	2.00	...	6.34	8.56	...	1.27	3.87	...	2.01
...	...	...	...	...	...	...	...	...	...
T_N	5.11	...	8.00	4.33	...	3.04	6.72	...	9.20

We labeled the data considering anomaly values for the five metrics exceeding the default thresholds (Table I). We labeled the anomalies with a boolean value, where 1 represents the data exceeding the threshold. Table I lists the five KPIs we considered as variables in this analysis, together with their respective thresholds, while the green points are normal values, instead the red points are anomalous values.

The result of the data collection are five csv files, which we used to train and test our Machine Learning algorithms. Table III shows an example of the data represented in one of the five csv files.

TABLE III
EXAMPLE OF LABELED DATA

	Dependent Variable	**Independent Variables**			
Time stamp	DM 1	M 1	M 2	...	M 167
T_1	1	1.27	3.87	...	2.01
...	...	...	...	...	...
T_N	0	4.57	3.86	...	y2.90

F. Data Analysis

We applied the eight algorithms described in Section II to verify whether there are dependencies between each dependent variables based on the independent ones. Then we compared their accuracy by means of the accuracy measures reported in Section III-B, (**RQ1**) as well as their training and test time (**RQ2**).

Moreover, in order to understand which metric contributes more to anomaly detection (**RQ3**), we also performed a Principal Component Analysis (PCA) and applied the drop-column algorithm.

Principal component analysis is a statistical algorithm that reduces data dimension while retaining most of the information by creating new components that summarize the data.

It is widely used in data mining for datasets investigation. In PCA, new orthogonal components (latent variables or principal components) are obtained by maximising the data variance. The total of the principal components (factors) is usually much lower than the total of original variables, so that the data can be visualised in a low-dimensional space. While principal components analysis decreases the space dimension, it does not decrease the number of the original components, as it uses all of them for the generation of the new latent variables (principal components). This aspect of PCA is leveraged to figure out the importance of features. In fact, features with the highest contribution to these components are the most important.

Drop-column mechanism[2] is a simplified alternative of the exhaustive search technique [22], which iteratively tests every subset of variables for their classification performance. The full exhaustive search is very time-consuming, because it requires 2X train-evaluation steps, where X is the dimension of the feature space. Instead, in the drop-column technique, individual features are dropped one at a time, instead of all possible groups of features. This means that a model is trained X times for a X-dimensional feature space, iteratively removing one feature at a time, from the first to the last of the data set. The difference in cross-validated test accuracy between the newly trained model and the baseline model (the one trained with the full set of features) defines the importance of that specific feature. The more the accuracy of the model drops, the more important is the specific feature for the classification. The importance of the metrics was not calculated for all the machine learning models described, but only for the most accurate model (cross-validated with all X features), because the feature importances of a classifier with lower accuracy performance were likely to be less reliable.

IV. Results

The data extracted from five weeks reported more than 800k rows, resulting in a 700MB csv file. Machine Learning algorithms were executed on a Linux Ubuntu virtual machine with 24 cores and 90GB RAM.

In the next sub-sections, we will report on the results obtained after analyzing the collected data.

RQ1: Is there a Machine Learning algorithm that can accurately detect performance-related anomalies in cloud-native systems?

All the accuracy measures adopted reported consistent results. We adopted the Receiver Operating Characteristics (ROC) and the related Area Under the Receiver Operating Characteristic Curve (AUC) for comparing the accuracy of the different models. For reasons of space, we only report results for AUC.

Only three variables can be predicted with an accuracy (AUC) higher than 90%, while two variables can be predicted with an accuracy (AUC) higher than 80%. The AUCs for each model and variable are reported in Table IV - VIII.

[2]https://explained.ai/rf-importance/

The comparison of the accuracy of the different Machine Learning models revealed that XGBoost is the most accurate model for four out of five KPIs, while in one case, ExtraTrees performed better than the others.

RQ2: Which Machine Learning algorithm can accurately detect performance-related anomalies with the shortest training time?

The training time of all the techniques except Logistic Regression was very short. After one week of training time, we stopped the execution of Logistic Regression, considering it too expensive to be applied in this context. For the other techniques, in some cases, some sub-optimal technique required a shorter training time than the optimal technique. For example, the testing time of Random Forest was much shorter than the one spent for training XGBoost. However, the difference is in the range of a few minutes. Since we are planning to train the systems once a week, we can consider the time differences as negligible. The comparison of the training time, testing time, and AUC is reported in Table IX.

TABLE IV
Accuracy metrics for % Network Processor Idling Time KPI (RQ1)

Classifier	TNR	FNR	FPR	Recall	Prec	f-meas	MCC
AdaBoost	96.16	82.67	3.842	17.33	11.95	10.60	11.84
DecisionTrees	92.43	70	7.566	30	22.71	13.86	15.79
ExtraTrees	99.05	96	0.95	4	1.18	1.82	2.14
GradientBoost	93.35	56.67	6.65	43.33	32.80	16.30	21.64
LogisticRegr.	87.56	88	12.44	12	0	0.01	-0.01
MLP	98.20	100	1.80	0	0	0	-0.04
RandomForest	98.79	91.33	1.21	8.67	4.12	3.90	4.84
XGBoost	99.96	83.33	0.03	16.67	23.15	10.94	14.41

TABLE V
Accuracy metrics for Request Queue Size KPI (RQ1)

Classifier	TNR	FNR	FPR	Recall	Prec	f-meas	MCC
AdaBoost	90.09	85.31	9.91	14.69	14.55	28.11	0.59
DecisionTrees	84.37	49.50	15.63	50.50	7.29	14.76	3.93
ExtraTrees	91.77	54.48	8.23	45.51	5.93	10.93	3.98
GradientBoost	82.10	50.20	17.90	49.80	7.34	14.01	3.35
LogisticRegr.	96.79	84.91	3.21	15.09	5.20	9.47	1.42
MLP	97.67	99.38	2.32	0.62	10.88	18.25	-0.36
RandomForest	91.72	69.32	8.28	30.68	5.91	10.09	2.47
XGBoost	98.05	86.89	1.95	13.11	3.51	6.99	1.33

TABLE VI
Accuracy metrics for Avg Request Latency KPI (RQ1)

Classifier	TNR	FNR	FPR	Recall	Prec	f-meas	MCC
AdaBoost	94.00	85.30	5.99	14.70	25.49	10.97	12.01
DecisionTrees	78.85	69.18	21.14	30.81	41.37	10.87	15.31
ExtraTrees	85.27	65.81	14.73	34.19	41.28	12.33	18.02
GradientBoost	80.33	79.86	19.67	20.14	21.26	2.97	3.37
LogisticRegr.	79.42	96.99	20.58	3.01	0.18	0.28	-4.20
MLP	99.89	94.78	0.11	5.22	3.28	4.03	4.05
RandomForest	92.28	65.54	7.72	34.46	51.96	24.27	29.63
XGBoost	93.92	85.88	6.08	14.11	22.83	10.88	11.16

TABLE VII
ACCURACY METRICS FOR MAX MESSAGE LAG KPI (RQ1)

Classifier	TNR	FNR	FPR	Recall	Prec	f-meas	MCC
AdaBoost	94.25	69.38	5.75	30.62	59.36	28.72	33.75
DecisionTrees	80.36	67.11	19.64	32.89	31.88	11.52	14.98
ExtraTrees	76.55	84.47	23.45	15.53	40.18	5.95	8.39
GradientBoost	88.37	64.87	11.63	35.12	37.49	19.62	23.50
LogisticRegr.	85.20	70.91	14.80	29.09	14.42	5.43	7.55
MLP	97.30	100	2.70	0	0	0	-0.37
RandomForest	80.14	71.93	19.86	28.07	40.45	11.42	14.32
XGBoost	99.23	57.17	0.77	42.82	67.36	41.30	46.73

TABLE VIII
ACCURACY METRICS FOR MIN FETCH RATE KPI (RQ1)

Classifier	TNR	FNR	FPR	Recall	Prec	f-meas	MCC
AdaBoost	95.14	77.91	4.86	22.09	66.78	23.20	30.85
DecisionTrees	88.49	78.67	11.52	21.32	34.95	11.45	15.70
ExtraTrees	89.51	92.16	10.49	7.83	54.33	4.01	9.83
GradientBoost	90.31	57.94	9.69	42.06	55.84	31.51	35.44
LogisticRegr.	87.76	81.25	12.24	18.75	26.67	7.21	9.93
MLP	98.00	99.91	2.00	0.09	0.01	0.01	-0.21
RandomForest	89.85	91.14	10.14	8.86	65.06	5.90	13.15
XGBoost	97.38	81.41	2.61	18.58	64.37	17.39	24.99

RQ3: What are the most important metrics to be considered when detecting performance-related anomalies?

The two methods adopted, Principal Component Analysis and drop-column mechanism (Table X), reported different results. This can be expected due to their different approaches. Below, the first ten most important metrics are reported for both the techniques. The blue lines represent the importance of each feature for the prediction of the respective KPI. The PCA reported the metric *"zookeeper-ElectionType"* as the most important predictor for all the five KPIs. Instead the drop-column algorithm reported that *"Server replica manager - underminisr partition count"* is the best predictor for % of network processor idling time, *"manual leader balance rate and time"* for max message lag, *"% of network processor idling time"* for min fetch rate, *"Request Queue Time"* for Avg request latency, and *"Number of Configuration Reload Failures"* for request queue size.

RQ4: How many components are necessary to accurately detect performance-related anomalies?

By performing the Principal Component Analysis for dimensional reduction on the data, it was possible to see how many components are necessary and how many can be removed for the prediction of the five KPIs.

Since the five charts produced are exactly identical, only one of them is reported in Figure 1. As stated in Section III-F, PCA generates new components that summarize the data. Often a part of the total components generated, are enough to describe the original data (more important). As it can be seen from Figure 1, out of a total of 168 components, 100 of them are enough to represent 95% of the original data. It is also possible to see that to have 100% of original data represented, we need 120 components. All the 120 metrics are reported in Prometheus in less than 200 milliseconds.

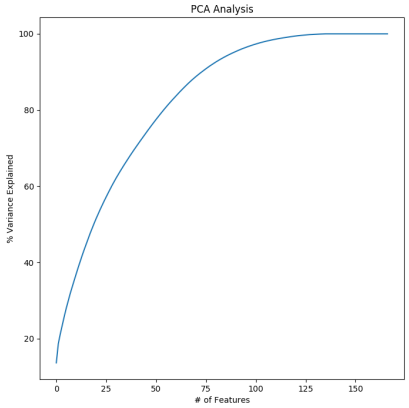

Fig. 1. Variance explained by the 168 KPIs (RQ4)

V. DISCUSSION

The application of the eight Machine Learning techniques on the data collected at runtime from the cloud-native system showed important dependencies between performance-related KPIs and the metrics collected in the system.

The most important outcome is that performance-related anomalies can be detected by monitoring a limited number of metrics collected at runtime. Some techniques require excessive training time (Logistic Regression). However, other techniques, such as XGBoost, provide a very high level of accuracy in four out of five cases with a brief training time. In the case of the Request Queue Size KPI, ExtraTrees algorithm performed better than the others.

The final result is that the number of components used for the training can be easily reduced to 120 preserving the same amount of information for the prediction of the anomalies. This result is important because by reducing the number of features, training and testing performances will improve.

The result of this study could help companies to understand how to monitor cloud-native systems and especially how to detect whether some KPIs they consider relevant are dependent on other metrics they can collect.

VI. THREATS TO VALIDITY

Construct Validity. We adopted the measures provided by Kafka, Zookeeper and Prometheus, since our goal was to understand dependencies between them instead of specifying existing metrics.

Internal Validity. Some metrics were duplicated, reporting the same value for the same service twice. We are aware of this issues and to reduce this threat, we considered only once duplicated data. We are also aware that we cannot claim a direct cause-effect relationship between the different metrics, also in case of positive correlations. Moreover, we are also aware that metrics with different roles (e.g., metrics of communication between services) can be more frequent than others.

External Validity. We are aware that different different systems might behave differently, neverthless we studied commonly used tools and metrics to allow for better generalization.

TABLE IX

ACCURACY AND TRAINING AND TESTING TIME COMPARISON FOR THE DIFFERENT MODEL USED AND FOR EACH KPI ANALYZED. THE AUC IS SHOWN IN TERMS OF MEAN AND STANDARD DEVIATION OF THE 10 FOLDS VALIDATION. THE TRAIN AND TEST TIME ARE SHOWN IN SECONDS.

KPI	Metric	XGBoost	AdaBoost	Log. Regr.	Rand. Forest	Grad. Boost	Dec.Tree	ExtraTree	MLP
Min Fetch Rate	AUC	0.96± 0.08	0.87± 0.17	0.72± 0.19	0.79± 0.24	0.67± 0.19	0.55± 0.13	0.65± 0.24	0.50± 0.02
	Training Time	645.71	1250.90	14574.06	45.22	1432.71	240.92	535.28	22712.68
	Testing Time	1.21	16.91	0.05	0.50	1.31	0.15	4.15	3.01
% Network Processor Idling Time	AUC	0.98± 0.04	0.86± 0.14	0.66± 0.25	0.83± 0.19	0.67± 0.18	0.61± 0.13	0.73± 0.25	0.51± 0.06
	Training Time	1552.42	2920.80	26550.41	54.20	3629.73	220.89	494.55	17911.42
	Testing Time	2.52	36.11	0.07	1.43	3.72	0.24	6.73	helmi.64
Request Queue Size	AUC	0.62± 0.18	0.58± 0.17	0.59± 0.19	0.86± 0.16	0.60± 0.14	0.67± 0.13	0.87± 0.15	0.48± 0.03
	Training Time	404.94	837.57	12612.92	33.57	1385.42	141.06	281.66	7766.97
	Testing Time	0.71	10.41	0.05	0.50	1.14	0.11	2.73	2.28
Avg Request Latency	AUC	0.84± 0.24	0.80± 0.25	0.60± 0.24	0.52± 0.28	0.75± 0.27	0.55± 0.17	0.59± 0.25	0.54± 0.09
	Training Time	1079.31	2096.02	24025.74	55.43	2992.22	267.68	571.29	14314.08
	Testing Time	2.12	24.61	0.06	1.52	3.06	0.23	7.35	4.17
Max Message Lag	AUC	0.96± 0.06	0.75± 0.22	0.69± 0.16	0.68± 0.25	0.62± 0.21	0.57± 0.14	0.50± 0.27	0.46± 0.05
	Training Time	779.93	1541.25	18015.86	109.86	3313.20	665.73	1192.65	22073.61
	Testing Time	1.56	17.31	0.05	1.33	3.14	0.30	7.55	3.54

TABLE X

MOST IMPORTANT METRICS EVALUATED BY THE DROP-COLUMN MECHANISM AND BY THE PRINCIPAL COMPONENT ANALYSIS. FOR THE FIRST IT IS SHOWN THE DROP IN OVERALL ACCURACY (IN PERCENTAGE). FOR THE LATTER IT IS INDICATED THE SCORE OF THE FIRST COMPONENT OF THE PCA

KPI	Technique	Best Feature	Importance (Score)
Min Fetch Rate	Drop-column	KAFKA NETWORKSOCKETSERVER NETWORKPROCESSORAVGIDLEPERCET	0.53%
	PCA	ZOOKEEPER ELECTIONTYPE	0.1905
% Network Processor Idling Time	Drop-column	KAFKA SERVER REPLICAMANAGER UNDERMINISRPARTITIONCOUNT	0.94%
	PCA	ZOOKEEPER ELECTIONTYPE	0.1892
Request Queue Size KPI	Drop-column	JMX CONFIG RELOAD FAILURE TOTAL	1.11%
	PCA	ZOOKEEPER ELECTIONTYPE	0.1871
Avg Request Latency	Drop-column	KAFKA NETWORK REQUESTMETRICS REQUESTQUEUETIMEMS	2.09%
	PCA	ZOOKEEPER ELECTIONTYPE	0.1871
Max Message Lag	Drop-column	KAFKA CONTROLLER CONTROLLERSTATS MANUALLEADERBALANCERATEANDTIMEMS	1.80%
	PCA	ZOOKEEPER ELECTIONTYPE	0.1871

Reliability. We do not exclude the possibility that other statistical or machine learning approaches, such as Deep Learning, might have yielded similar or even better accuracy than our modeling approach. In our case, this threat could be represented by the analysis method applied in our study. We reported the results considering descriptive statistics. Moreover, instead of using only Logistic Regression, we compared the prediction power of different classifier to reduce the bias of the low prediction power that one single classifier could have. We do not exclude the possibility that other statistical or machine learning approaches such as Deep Learning or others might have yielded similar or even better accuracy than our modeling approach. However, considering the extremely low importance of each TD Issue and its statistical significance, we do not expect to find big differences applying other type of classifiers.

VII. RELATED WORK

Anomaly detection has been investigated in several domains in recent years by applying probabilistic [23] and statistical [24] approaches.

Hochenbaum et al. [25] developed two statistical approaches for anomaly detection in cloud infrastructure data. Their first method called Seasonal-ESD combines seasonal decomposition and the Generalized ESD test, for anomaly detection. The second approach called Seasonal-Hybrid-ESD (S-H-ESD) adds statistical measures such as median and median absolute deviation (MAD) to the previous algorithm. Both methods perform quite well, achieving an F-measure between 0.96 and 0.98 on injected anomalies.

Solaimani et al. [26] proposed a Chi-square based anomaly detection approach on heterogeneous data by leveraging the high processing power of Apache Spark. The results obtained showed that with their methods, the authors were able to obtain a TPR of 90% and FPR of 1.2%,

Smrithy et al. [27] developed an algorithm based on Kolmogorov-Smirnov goodness of fit test for anomaly detection of access requests at runtime in cloud enviroments.However, they did not provide information on the accuracy of their anomaly detection model.

Wang et al. [28] proposed statistical techniques for online anomaly detection. The proposed approaches are lightweight and based on Tukey and Relative Entropy statistics. Their approach was tested for multi-tier web application running on server class machine. They were able to detect $\sim 86\%$ of anomalies with a FPR of 0.04.

Roy et al. [29] developed PerfAugur, a system for the detection of anomaly behaviors using data mining algorithms in service logs. The achieved a F1 score of ~ 0.80 in for synthetic data and more than 0.98 for real data.

Statistical models perform well in identification of anomalies and they do not require a big amount of data for training models. Despite this, the main obstacle of these techniques is the production of biased results in case of inaccurate hypothesis on the data. This leads to many false positives and makes statistical approaches not suitable for real applications.

On the other hand, machine learning approaches are capable of inferring distribution of normal and anomalous behaviors, and determine anomalies by using supervised, semi-supervised, unsupervised, or deep learning techniques [30].

Ahmed et al. [31] proposed a sequential anomaly detection technique based on the kernel version of the recursive least squares algorithm. This approach can be used effectively also for multivariate data. This approach was applied for anomalies

in network traffic, and the authors found that for the specific case, their approach works better than other offline methods like PCA and One-Class Nieghbour Machine (OCNM). This method was able to detect between 28 and 39 anomalies out of 44 present in the Abilene Flow-Counts Dataset, and between 21 and 30 out of 34 anomalies present in the Abilene Packet-Counts Dataset.

Lakhina et al. [32] presented an anomaly detection approach based on the division of the high-dimensional space represented by a set of metrics into disjoint subspaces corresponding to normal and anomalous behaviors. To perform the separation, Principal Component Analysis has been employed successfully. The approach was applied also in this case to network traffic in order to identify volume anomalies. The results showed that through their method, it is possible to achieve a high detection rate of about 90%, while maintaining a low false alarm rate.

Ibidunmoye et al. proposed two methods, PAD [33] and BAD [34], based on statistical analysis and kernel density estimation (KDE) applied to unbalanced data. The performances of these methods are affected by the window size used for the estimation. Both these methods result efficient in finding anomalies with low false positive rate, in network traffic. In fact, they could achieve an average false positive rate of 8.7% for PAD and 4.4% for BAD.

Thill et al. [35] proposed SORAD, a simple anomaly detection approach based on regression techniques. This was applied on Yahoo Webscope S5 benchmark, showing remarkably good result, although being a simple algorithm compared to others. It achieved on F1 score between 0.64 and 0.99 on the four dataset tested.

Ahmad et al. [36] presented a real-time anomaly detection algorithm based on Hierarchical Temporal Memory (HTM) and suitable for spatial and temporal anomaly detection in predictable and noisy environments. This was tested on Numenta Anomaly Benchmark (NAB), which contains data streams with labeled anomalies, performing better than other classic methods. The metric used for assessing the technique performance is the NAB score. This is a function designed withing the NAB dataset which gives positive point for earlier detection and negative points for later.

Mi et al. [37] developed CloudDiag, a tool for performance anomaly detection based on unsupervised learning, in order to detect performance anomalies in production cloud computing system. This system perform significantly better than the PCA-approach tested as comparison, scoring precision and recall greater than 90% on all case tested in the study.

Dean et al. [38] developed UBL, a distributed and scalable anomaly detection system for Infrastructure as a Service (Iaas) cloud environments based on unsupervised learning. Leverages the power of Self-Organizing Map (SOM) to detect performance-related anomalies to provide suggestions on possible issues. This approach allows to detect more than 90% of the anomalies in most of the cases tested.

Tan et al. [39] developed PREPARE, a performance-related anomaly prevention system for virtualized cloud computing infrastructure. It combines attribute value prediction with supervised methods to perform resource scaling for performance anomalies prevention. This system allows to detect anomalies between $\sim 80\%$ and $\sim 99\%$ of the cases.

Guan et al. [40] implemented an unsupervised proactive feature management framework for cloud infrastructures based on a combination of Bayesian models to perform anomaly detection with high true positive rate and low false positive rate. No clear accuracy metrics were reported in the paper.

Gulenko et al. [41] proposed an event-based approach to real-time anomaly detection in cloud-based systems with a specific focus on the deployment of virtualized network functions. They applied both supervised and non-supervised classification algorithms, obtaining good results in the identification of anomalies, which were identified in 95% of the cases.

Monni et al. [6] proposed an energy-based anomaly detection tool (EmBeD) for the cloud domain. The tool is based on a Machine Learning approach, based on an RBM, and is able to reveal failure-prone anomalies at runtime. EmBeD exploits the system behavior using the raw metric data, classifying the relationship between anomalous behavior and future failures with a good level of accuracy (in terms of very few false positives). Moreover, Monni et al. [6] also defined an energy-based model to capture failure-prone behavior without training with seeded errors. They identified important analogies regarding the nature of complex software systems, complex physical systems, and complex networks. The results obtained show that their methods can achieve a f1-score $\sim 95\%$ to $\sim 98\%$

Sauvanaud et al. [5] applied machine learning approaches such as Neural Networks, Naive Bayes, Nearest Neighbors, and Random Forest for anomaly detection at metric level. The anomaly detection system presented in their work scored an AUC of $\sim 90\%$.

VIII. Conclusion

The application of the eight Machine Learning techniques on the data we collected at runtime from our cloud-native system showed important dependencies between performance-related KPIs and the metrics collected in the system.

The most important outcome is that it is possible to detect performance-related anomalies by monitoring a limited number of metrics collected at runtime. Some techniques require too much training time (Logistic Regression). However, other techniques such as XGBoost provide a very high level of accuracy in four out of five cases with a very short training time. As for the metrics that can be used to predict the anomalies, the PCA reported the metric *"zookeeper-ElectionType"* as the most important predictor for all the five KPIs. Instead the drop-column algorithm reported that *"Server replica manager - underminisr partition count"* is the best predictor for % of network processor idling time, *"manual leader balance rate and time"* for max message lag, *"% of network processor idling time"* for min fetch rate, *"Request Queue Time"* for Avg request latency and *"Number of Configuration Reload Failures"* for request queue size.

We are aware of several limitations of this work. We do not exclude the possibility that other statistical or Machine Learning approaches might have yielded similar or even better accuracy than our modeling approach. Moreover, our study aimed to identify dependencies between the five KPIs identified and the metrics collected at runtime. We do not exclude the possibility that other metrics might predict one of these KPIs better, but at the moment, we are not allowed to change the configuration of the monitoring systems and to add more metrics. The result of this study could help other companies understand how to monitor cloud-native systems and especially how to detect whether some KPIs they consider relevant are dependent on other metrics they can collect.

Future work will include the application of different techniques, such as time series analysis so as to understand how early it is possible to predict anomalies, and which is the algorithm that can be effectively used in this context. Moreover, we are planning to investigate suitable techniques for predicting the fault-proneness of the different metrics so as to be able to react on time, before the anomaly happens. We are also planning the development of a set of plug-ins for Prometheus to graphically represent the results in dashboards.

REFERENCES

[1] X. Chen, C. Lu, and K. Pattabiraman, "Failure analysis of jobs in compute clouds: A google cluster case study," in *International Symposium on Software Reliability Engineering*, Nov 2014, pp. 167–177.

[2] F. Lomio, D. M. Baselga, S. Moreschini, H. Huttunen, and D. Taibi, "Rare: a labeled dataset for cloud-native memory anomalies," in *International Workshop on Machine-Learning Techniques for Software-Quality Evaluation*, 2020, pp. 19–24.

[3] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth, "Performance anomaly detection and bottleneck identification," *ACM Comput. Surv.*, vol. 48, no. 1, pp. 4:1–4:35, Jul. 2015.

[4] S. Jin, Z. Zhang, K. Chakrabarty, and X. Gu, "Changepoint-based anomaly detection in a core router system," in *International Test Conference (ITC)*, Oct 2017, pp. 1–10.

[5] C. Sauvanaud and et al., "Anomaly detection and diagnosis for cloud services: Practical experiments and lessons learned," *Journal of Systems and Software*, vol. 139, pp. 84 – 106, 2018.

[6] C. Monni, M. Pezzè, and P. Gaetano, "An rbm anomaly detector for the cloud," in *International Conference on Software Testing*, 2019.

[7] F. Lomio, S. Jurvansuu, and D. Taibi, "Metrics selection for load monitoring of service-oriented system," in *International Workshop on Machine Learning Techniques for Software Quality Evolution*, 2021.

[8] M. Fowler and J. Lewis, "Microservices," 2014. [Online]. Available: http://martinfowler.com/articles/microservices.html

[9] S. Newman, *Building Microservices*. O'Reilly Media, Inc., 2015.

[10] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[11] D. R. Cox, "The regression analysis of binary sequences," *Journal of the Royal Statistical Society. Series B*, vol. 20, no. 2, pp. 215–242, 1958.

[12] L. Breiman, J. Friedman, C. J. Stone, and R. Olshen, *Classification and regression trees Regression trees*, 1984.

[13] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 8 1996.

[14] ——, "Random forests," *Machine learning*, vol. 45, 2001.

[15] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, 4 2006.

[16] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 8 1997.

[17] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, pp. 1189–1232, 2000.

[18] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," 2016.

[19] D. M. W. Powers, "Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation," *Journal of Machine Learning Technologies*, vol. 2, no. 1, pp. 37–63, 2011.

[20] K. Hajian-Tilaki, "Receiver operating characteristic (roc) curve analysis for medical diagnostic test evaluation," *Caspian journal of internal medicine*, vol. 4, no. 2, p. 627, 2013.

[21] F. Pedregosa and et al., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[22] H. Yoon, K. Yang, and C. Shahabi, "Feature subset selection and feature ranking for multivariate time series," *IEEE transactions on knowledge and data engineering*, vol. 17, no. 9, pp. 1186–1198, 2005.

[23] Z. Guo, G. Jing, H. Chen, and K. Yoshihira, "Tracking probabilistic correlation of monitoring data for fault detection in complex systems," in *Int. Conf. on Dependable Systems and Networks*, 2006, pp. 259–268.

[24] T. Idé and H. Kashima, "Eigenspace-based anomaly detection in computer systems," in *Int. Conf.on Knowledge Discovery and Data Mining*, 2004.

[25] J. Hochenbaum, O. Vallis, and A. Kejariwal, "Automatic anomaly detection in the cloud via statistical learning. eprint," *arXiv*, 2017.

[26] 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 *International Conference on Big Data (Big Data)*. IEEE, 2014, pp. 1086–1094.

[27] G. Smrithy and R. Balakrishnan, "A statistical technique for online anomaly detection for big data streams in cloud collaborative environment," in *International Conference on Computer and Information Technology (CIT)*. IEEE, 2016, pp. 108–111.

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

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

[30] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth, "Performance anomaly detection and bottleneck identification," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, p. 4, 2015.

[31] T. Ahmed, M. Coates, and A. Lakhina, "Multivariate online anomaly detection using kernel recursive least squares," in *IEEE International Conference on Computer Communications*. IEEE, 2007, pp. 625–633.

[32] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *ACM SIGCOMM computer communication review*, vol. 34, no. 4. ACM, 2004, pp. 219–230.

[33] O. Ibidunmoye, T. Metsch, and E. Elmroth, "Real-time detection of performance anomalies for cloud services," in *International Symposium on Quality of Service (IWQoS)*. IEEE, 2016, pp. 1–2.

[34] O. Ibidunmoye, A.-R. Rezaie, and E. Elmroth, "Adaptive anomaly detection in performance metric streams," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 217–231, 2017.

[35] M. Thill, W. Konen, and T. Bäck, "Online anomaly detection on the webscope s5 dataset: A comparative study," in *Evolving and Adaptive Intelligent Systems (EAIS)*. IEEE, 2017, pp. 1–8.

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

[37] H. Mi, H. Wang, Y. Zhou, M. R.-T. Lyu, and H. Cai, "Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1245–1255, 2013.

[38] D. J. Dean, H. Nguyen, and X. Gu, "Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems," in *International conference on Autonomic computing*. ACM, 2012, pp. 191–200.

[39] Y. Tan and et al., "Prepare: Predictive performance anomaly prevention for virtualized cloud systems," in *International Conference on Distributed Computing Systems*. IEEE, 2012, pp. 285–294.

[40] Q. Guan, Z. Zhang, and S. Fu, "Ensemble of bayesian predictors for autonomic failure management in cloud computing," in *International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2011, pp. 1–6.

[41] A. Gulenko, M. Wallschläger, F. Schmidt, O. Kao, and F. Liu, "A system architecture for real-time anomaly detection in large-scale nfv systems," *Procedia Computer Science*, vol. 94, pp. 491 – 496, 2016.