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. 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, for example 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. Services are usually running very closely to the maximum hardware capacity, especially when running on private cloud systems, therefore, predicting performance-related 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 [3], allowing researchers to propose both supervised and unsupervised Machine Learning (ML) techniques for anomaly detection. Supervised models train the model under consideration of both normal and failing execution data [4] [5], while unsupervised are based only on normal execution time at the price of a reduced accuracy.

Besides accuracy, the time required by ML 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 ML technique might be most suitable for our purpose, we compared eight different ML 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.

II. Empirical Study Design

The goal of this work is to analyze eight ML 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 ML algorithm that can accurately detect performance-related anomalies in cloud-native systems?

RQ2. Which ML 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?

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.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

Accuracy. To assess the detection accuracy of the different ML 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. As accuracy metrics, we first calculated precision and recall. However, as suggested by [8], 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 and the related f-measure help to overcome this issue. Moreover, as recommended by Powers [8], 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. Finally, we calculated the Receiver Operating Characteristics (ROC) and the related Area Under the Receiver Operating Characteristic Curve (AUC).

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.

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.

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.

We labeled the data considering anomaly values for the five metrics exceeding the default thresholds (Table **??**). We labeled the anomalies with a boolean value, where 1 represents the data exceeding the threshold. Table **??** 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 ML algorithms.

Data Analysis. We applied the eight algorithms 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, (**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. Drop-column mechanism[2] is a simplified alternative of the exhaustive search technique [9], which iteratively tests every subset of variables for their classification performance.

III. Results and Discussion

RQ1. 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 I. The comparison of the accuracy of the different ML 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. 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 the online appendix.

RQ3. The two methods adopted, Principal Component Analysis and drop-column mechanism (Table II), 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.

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

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

TABLE I

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

RQ4. 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 **??**. 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). 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.

Discussion. ML techniques 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.

IV. Threats to 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. 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 can be more frequent. We are aware that different different systems might behave differently, nevertheless we studied commonly used tools and metrics to allow for better generalization. We do not exclude the possibility that other statistical or ML approaches, such as Deep Learning, might have yielded similar or even better accuracy than our modeling approach.

V. Related Work

Anomaly detection has been investigated in several domains in recent years by applying probabilistic [10] and statistical [11] approaches. Statistical models [12] [13] [14], 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, ML approaches are capable of inferring distribution of normal and anomalous behaviors, and determine anomalies by using supervised [15], semi-supervised, unsupervised [16] [17] [18], or deep learning techniques [19]. Lakhina et al. [20] 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.

Ibidunmoye et al. proposed two methods, PAD [21] and BAD [22], based on statistical analysis and kernel density estimation (KDE) applied to unbalanced data. Thill et al. [23] 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. [24] 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.

Gulenko et al. [25] proposed an event-based approach to real-time anomaly detection in cloud-based systems with a specific focus on the deployment of virtualized network

TABLE II

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

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 results obtained show that their methods can achieve a f1-score $\sim 95\%$ to $\sim 98\%$

VI. Conclusion

The application of the eight ML 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 XG-Boost provide a very high level of accuracy in four out of five cases with a very short training time. 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.

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] 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.

[9] 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.

[10] 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.

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

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.