EPJ Web of Conferences **251**, 02011 (2021) https://doi.org/10.1051/epjconf/202125102011
CHEP 2021

Anomaly detection in the CERN cloud infrastructure

Domenico Giordano[1,*], *Matteo* Paltenghi[1], *Stiven* Metaj[1], and *Antonin* Dvorak[2]

[1]CERN, Geneva, Switzerland
[2]Nuclear Physics Institute of the Czech Academy of Sciences, Řež, Czech Republic

Abstract. Anomaly detection in the CERN OpenStack cloud is a challenging task due to the large scale of the computing infrastructure and, consequently, the large volume of monitoring data to analyse. The current solution to spot anomalous servers in the cloud infrastructure relies on a threshold-based alarming system carefully set by the system managers on the performance metrics of each infrastructure's component. This contribution explores fully automated, unsupervised machine learning solutions in the anomaly detection field for time series metrics, by adapting both traditional and deep learning approaches. The paper describes a novel end-to-end data analytics pipeline implemented to digest the large amount of monitoring data and to expose anomalies to the system managers. The pipeline relies solely on open-source tools and frameworks, such as *Spark*, *Apache Airflow*, *Kubernetes*, *Grafana*, *Elasticsearch*. In addition, an approach to build annotated datasets from the CERN cloud monitoring data is reported. Finally, a preliminary performance of a number of anomaly detection algorithms is evaluated by using the aforementioned annotated datasets.

1 Introduction

To accomplish its research goals in High Energy Physics, CERN relies on the computational power of its data centre. The computing facility is structured as a private cloud [1] managed via components of *OpenStack* [2], the free open-standard Infrastructure-as-a-Service platform. At the time of writing, the CERN *OpenStack* cloud contains about 8 000 bare-metal servers, configured as hypervisors hosting about 35 000 virtual machines [3]. Those virtual machines run either high-throughput computations for the various physics experiments or applications and services for the whole CERN campus. In both cases, a malfunctioning of the infrastructure's components can degrade the quality of service for CERN users. For this reason, CERN invests considerable amount of resources in making sure that its IT services are redundant, appropriately monitored and that self-healing procedures are implemented when possible. The status of the cloud infrastructure is inspected via an abundant number of dashboards that expose monitoring data and alarms' reports to the service managers. The alarming systems mainly adopt threshold-based triggers that capture only well known issues and, sometimes, are prone to generate false alarms and to overwhelm the service managers with unneeded notifications.

In this work an alternative approach to anomaly detection for computing facilities is proposed. Because of the lack of enough labeled data, the approach is based on unsupervised

*e-mail: domenico.giordano@cern.ch

machine learning techniques and targets time series, being this the intrinsic nature of most of the collected monitoring metrics. Section 2 details the application area and introduces a mathematical formulation of the problem. Section 3 addresses the data handling and computational challenges, as well as the implemented solutions. The preparation of a field-specific annotated dataset is covered in Section 4, and a proof of the system in action is given in Section 5.

2 Anomaly detection use case

In the current operation of the *OpenStack* cloud at CERN, the main usage of its central monitoring infrastructure [4] concerns the real-time inspection of the data center status, the threshold-based alarming on the infrastructure components, and the post-mortem analysis of issues after notifications by service's users. The adopted threshold-based alarm mechanism is in part embedded in the centralised monitoring dashboards and in part deployed in a number of sensors running in each bare-metal server and virtual machine.

With the exception of service logs, the monitored quantities have a numerical value, the production timestamp and several tags related to the producer entity. Therefore to identify misbehaving servers with anomaly detection approaches, logical candidates are the forecasting techniques applied to multivariate time series produced by a given server.

Another distinctive attribute for servers in a data centre is their organization in groups of hosts (a.k.a. hostgroups) having same configuration, same designation scope and often same topological placement in the data centre. Therefore servers in the same hostgroup have, on average, similar performance, and the deviations from the ensemble behavior can be used to spot anomalies. These are the fundamental assumptions on which the algorithmic strategy has been designed as described in the following problem formulation.

2.1 Problem formulation

Given an entity h, a server in this context, identified by a k-dimensional data point $\vec{m}(h) \in R^k$ in the space of the k monitored metrics of that entity, the goal of the anomaly detection is to model a binary prediction function $AD(\vec{m}(h)) : R^k \rightarrow \{0, 1\}$ such that $AD(\vec{m}(h)) = 1$ predicts h is anomalous and $AD(\vec{m}(h)) = 0$ predicts h is normal. This is usually achieved through a two-steps procedure: at first an anomaly score is evaluated for each entity, using a scoring function $s(\vec{m}(h)) : R^k \rightarrow R$. The goal of a machine learning approach is to model or learn this scoring function. Then, a step function $T_c(s) : R \rightarrow \{0, 1\}$ is applied to make the prediction binary, and identify the two classes normal (0) and anomalous (1). The threshold c is in general chosen fixing a maximum desired false-alarm rate, measured on a validation dataset.

In the case of time series metrics, the temporal dimension needs to be included, and the sequence of t consecutive measurements $\{\vec{m}_1(h), \vec{m}_2(h), ..., \vec{m}_t(h)\}$ will represent the evolution of the given entity in the space of the monitored metrics. The server metrics are typically collected at intervals of time in the order of seconds or minutes. This fine-grained time resolution is not essential if the goal of the anomaly detection is the identification of anomalies existing for a sizable amount of time (hours) respect to the identification of any punctual anomaly. Therefore a data reduction is typically applied to the original time series, aggregating the measurements within a coarser time resolution (r) and extracting summary statistics. In the simplest scenario, the summary statistic is the average, and the time gap between data points at time t and $t + 1$ is in the order of tens of minutes.

Moreover, in order to identify long-living anomalies, the time series metrics are analised in windows of w consecutive data points, and the anomaly score of a given server h is determined for the whole window $W^k_{i,w}(h) = \{\vec{m}_i(h), \vec{m}_{i+1}(h), ..., \vec{m}_{i+w-1}(h)\}$ at once, i.e. the binary

prediction function applies to the R^{k*w} space $AD(W^k_{i,w}(h)) : R^{k*w} \rightarrow \{0, 1\}$. For this purpose various anomaly detection algorithms are compared in Section 5.

The set of data $W^k_{i,w}(h)$ is usually represented as a matrix of k metric-rows and w time-columns, where each metric value m^j has been standardized over a standardization time window T_z:

$$W^k_{i,w}(h) = \begin{pmatrix} m^1_i & m^1_{i+1} & \cdots & m^1_{i+w-1} \\ m^2_i & m^2_{i+1} & \cdots & m^2_{i+w-1} \\ \vdots & \vdots & \ddots & \vdots \\ m^k_i & m^k_{i+1} & \cdots & m^k_{i+w-1} \end{pmatrix} \tag{1}$$

This matrix representation, named "Multivariate Temporal Window", is suitable for algorithms that can model the temporal dimension, as in the case of several deep learning algorithms. It has also a simple visual representation used to inspect data with a heatmap plot (Fig. 1). For those algorithms that are not able to model the temporal dimension, a different data representation is used, by flattening the matrix into a vector and extending the feature space from k to $k * w$. Since there is not anymore a direct representation of time, part of the correlation information is lost (or more convoluted) in this representation, although the two representations contain the same amount of data. The flattened representation is used mainly for traditional machine learning methods (see Sec. 5).

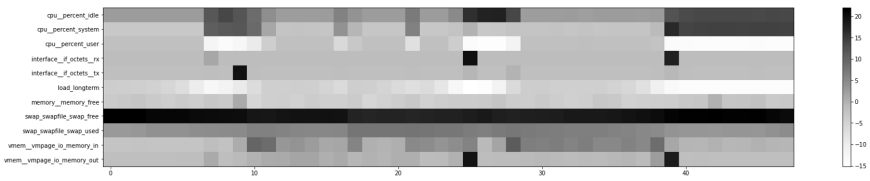


Figure 1. Heatmap representation, in grey tones, of a standardized Multivariate Temporal Window for a single server. On the y-axis, the labels identifying the 11 used metrics.

2.2 Dataset definitions

In order to fit and test the performance of the implemented algorithms, a training and a test dataset are used. The training dataset D_{train} is built as the union, $\bigcup$, of consecutive Multivariate Temporal Windows (CW), each of length w, inherent to the group (G) of servers being in the same hostgroup. The test dataset D_{test} follows a similar definition, but requires non-overlapping Multivariate Temporal Windows ($NOCW$) that come immediately after the training set:

$$D_{train} = \bigcup_{i \in CW} \bigcup_{h \in G} \{W^k_{i,w}(h)\} \qquad D_{test} = \bigcup_{i \in NOCW} \bigcup_{h \in G} \{W^k_{i,w}(h)\} \tag{2}$$

Although the choice of parameters such as the windows length (w), the reduction of resolution (r) and the size of CW could be the outcome of an optimization search, a more pragmatic approach has been followed in this phase of the study, and has been based on data availability, domain-expert knowledge and expectations. Therefore r has been fixed to 10 minutes, in order to remove the high-frequency components of the signals, and make the system more robust towards false alarms due to values' fluctuations. The windows length w has been fixed

to 8 hours, so that the daily data are split in 3 consecutive non-overlapping windows. The overall size of both *CW* and *NOCW* has been fixed to 1 week, considering it enough to learn a model of normality.

3 Anomaly detection pipeline

The primarily purpose of this work has been the design and implementation of a flexible and scalable anomaly detection system beneficial to any CERN service. The cloud service has been taken as initial concrete use case, and a number of anomaly detection algorithms suitable for this use case have been also evaluated. In order to be flexible, the system has been made modular, using the *Docker*[1] container technology to encapsulate each phase of the data-analytics pipeline. This choice allows also the system to scale out, parallelising the processing on the basis of the input data and the number of running algorithms. Another requirement imposed in the system design, has been the integration with the CERN monitoring infrastructure to ease the user adoption. This implies accessing the input data from the available data silos and exposing the anomaly detection results via the same monitoring dashboards that the experts use daily to monitor the cloud infrastructure.

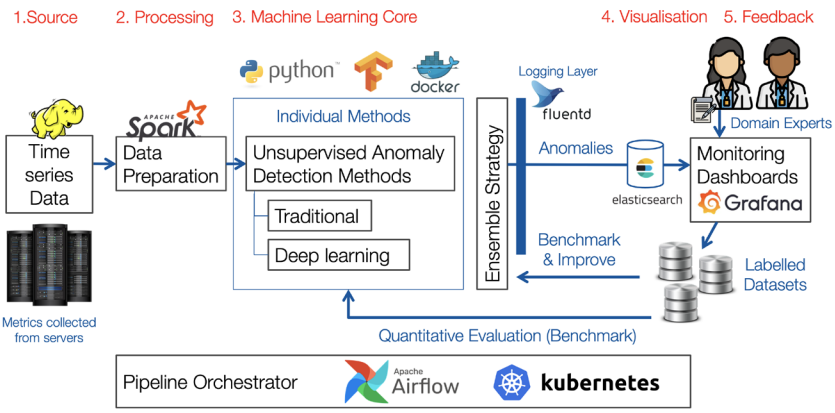


Figure 2. Anomaly detection pipeline with expert feedback loop. Note that the experts can see the result of the algorithms directly in the monitoring interactive dashboards, used also to inject validation labels. (Image drawn using resources from Flaticon.com)

Figure 2 shows a schema of the anomaly detection system, with emphasis on the different phases of the pipeline, and the main technologies adopted in each phase. The input time series are stored at CERN in *Hadoop Distributed File System* and can be accessed via *Apache Spark* batch processing. The first phase of the pipeline is called "data preparation". Deploying a dedicated container with Spark-client libraries, the anomaly detection system is connected to the Spark batch cluster and executes the pre-processing tasks: data extraction, time-resolution reduction, standardization, conversion into the Multivariate Temporal Window data format, data transfer to the storage suitable for the next phase.

The next phase is the anomaly detection core activity: each configured algorithm runs in separate containers spawn in a *Kubernetes* cluster. This choice allows the processing to

[1]NB: Widely known software packages and tools are reported in *Italic* and not referenced in the bibliography, being easily discoverable by any web search.

Figure 3. Grafana dashboard reporting candidate anomalies identified by the anomaly detection system.

be parallelised on the number of algorithms and the number of hostgroups. The anomaly detection results feed "ensemble engines" that produce additional anomaly scores based on ensemble techniques to provide a more robust anomaly detection.

Every container running an anomaly detection algorithm produces at the end an anomaly report, in *json* format. This report is injected in the CERN monitoring infrastructure using the data insertion mechanism for logging data, based on *fluentd*. The report includes details that uniquely identify a given run, the algorithm's configuration, the application's exit status and the list of entities identified as anomalous with their scores. The document is stored in an *Elasticsearch* database, and can be visualized in dedicated *Grafana* dashboards allowing user-friendly inspection (Fig. 3).

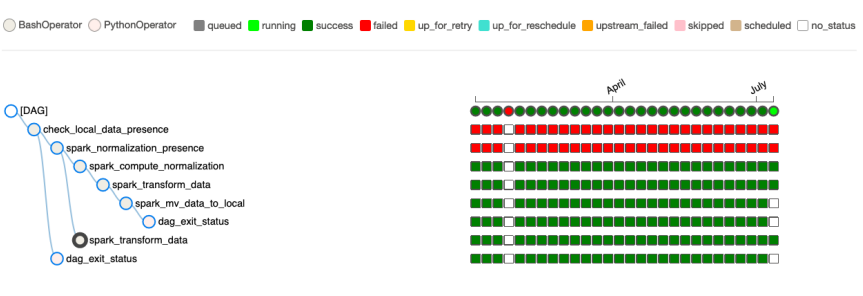


Figure 4. Summary monitoring of a recurrent anomaly detection pipeline, implemented as *Airflow* DAG. The status of each DAG component is reported with different color codes.

The anomaly detection pipeline has to run periodically in order to examine new time intervals, and can be configured with different algorithms for different input datasets. The role of orchestrating the pipeline execution and monitoring the running status is filled by *Apache Airflow*, an open-source workflow management platform. It has been chosen due to its flexibility in supporting different technologies, including the ones adopted in our project, such as *Docker* and *Kubernetes*, via the operator concept [5]. In addition *Airflow* is easily deployed starting from a *Docker-compose* configuration, and offers a user web interface as well as embedded monitoring of the planned tasks. These tasks are designed via directed acyclic graphs

(DAG), implemented in *Python* code. Figure 4 shows a typical anomaly detection pipeline implemented as *Airflow* DAG and executed multiple times in a cronjob-like mode. Similarly, the test and performance study of each adopted algorithm is automated via dedicated DAGs, that take care of pre-processing the testing data accordingly to the associated training data, then run the performance analysis. A summary view of the components and services deployed to run the anomaly detection pipeline is shown in Figure 5. The code is available at [6].

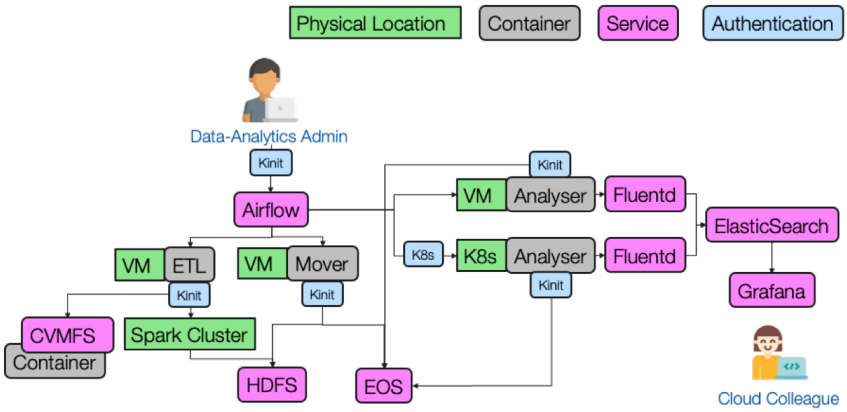


Figure 5. Schema of the anomaly detection system with emphasis on the services and technologies adopted in each phase.

4 Annotated datasets

When multiple machine learning algorithms are evaluated to select the best one in achieving a given goal, a crucial component of the activity is the definition of performance indicators. For anomaly detection algorithms, where the objective is to discriminate between two classes, indicators such as sensitivity, specificity and accuracy are used. These indicators require the availability of an annotated dataset, where data have additional labels indicating the true category they belong to. In the domain of anomaly detection for time series, the available annotated datasets are limited to relatively simple benchmark scenarios [7], often involving univariate metrics or signals not related to the problem under exam in this work, i.e. the anomaly detection for computing facilities.

With the intent of complementing this work with benchmarks studies based on a realistic annotated dataset, the authors propose a procedure to build such dataset using the actual CERN cloud monitoring data. This procedure involves the contribution of the CERN cloud experts to annotate, during their daily monitoring duties, the servers' status as normal or anomalous. This concept is represented in figure 2 by the domain experts inserting their annotations via *Grafana* dashboards. The annotation are then stored in a dedicated data base, ready to be re-injected in the pipeline workflow through a feedback loop.

The choice of *Grafana* as human-machine interface to annotate data is based on two facts: (i) the *Grafana* platform is well known and widely used by the CERN system managers, therefore would not represent an additional tool to learn and to support, and could possibly lead to a faster adoption of annotation practices by the CERN experts; (ii) *Grafana* already provides a basic functionality to manually insert annotations [8] in the dashboards. These

annotations support both point and time-interval annotation on the displayed time series data. The annotations are stored in the internal *Grafana* database and can be retrieved and displayed in the dashboards via tag-based queries.

4.1 Extension of Grafana annotation tool

Figure 6. Modification introduced in the *Grafana* annotation interface: “Normal” and “Anomaly” buttons are available in the annotation input form, and the dashboard’s template-variables are automatically appended as annotation’s tags.

Although the *Grafana* annotation tool comes already with useful core features, in order to adopt it to build a labelled dataset, additional properties have been implemented in what is referred as an extension of the *Grafana* tool. The extension requires only a minor modification [9] of the client-side *JavaScript* code running in the user’s browser, and introduces two features (Fig. 6): firstly it extends the annotation form to expose two additional buttons with label “Normal" and “Anomaly”, that, when clicked by the user, include automatically an additional tag to the annotation being created, with tag value equal to the button label. This approach enforces the naming convention for the two categories of interest and avoids human arbitrariness.

The second feature is related to the need of associating a given annotation with the specific server being displayed by the dashboard when the annotation is created. Most of the adopted *Grafana* dashboards widely use the template-variables mechanism to modify the dashboard data content on the basis of a set of tags displayed via drop-down menu in the dashboard’s header. Therefore, it is necessary to include in the stored annotation the tag values of all the template-variables currently used by the dashboard. This procedure is tedious and error prone if done manually by the user, but can be easily implemented modifying the *JavaScript* code of the annotation form.

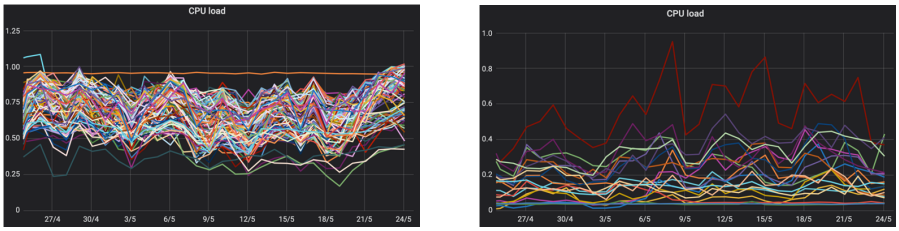
Ideally the extension should be eventually deployed at the server side but, as quick alternative, can be deployed in the user’s browser using simple local override browser extensions. Being this modification relevant in all those cases in which the annotations depend not only on the dashboard but also on the values of the template-variables that modify the visualised

data, a discussion with the *Grafana* upstream community is in action for the generalization of the extension for other use cases[2].

5 Anomaly detection system in action

The end-to-end usability of the implemented anomaly detection system has been evaluated by processing the monitoring metrics of a subset of about 300 CERN's servers, and running a number of anomaly detection algorithms that have been then benchmarked.

The CERN computational needs can be summarised in two main categories: (i) high-throughput computations for the various physics experiments or (ii) applications and services for the whole CERN campus (e-mail services, web services, database service, interactive clusters, personal virtual machines, etc.). The first category deploys clusters of virtual machines executing HTCondor [10] batch jobs. The bare-metal servers hosting these workloads are exclusively allocated for them, and represent 75% of the CERN computing capacity. The second category is allocated in other bare-metal servers configured to host virtual machines from multiple services in a shared resource fashion. Those services are less computational intensive, therefore the hosting servers are over-committed in terms of allocated virtual CPUs respect to the servers' CPUs. The different servers' configurations and purposes for the two use cases, determine different patterns in the monitored metrics (Fig. 7) and motivate their separate analysis in two distinct datasets, here referred as "Batch" and "Shared".



(a) "Batch" user category (b) "Shared" user category

Figure 7. Typical CPU load patterns for servers in the two main user categories, "Batch" (a) and "Shared" (b). Each line represents the CPU load of an individual server.

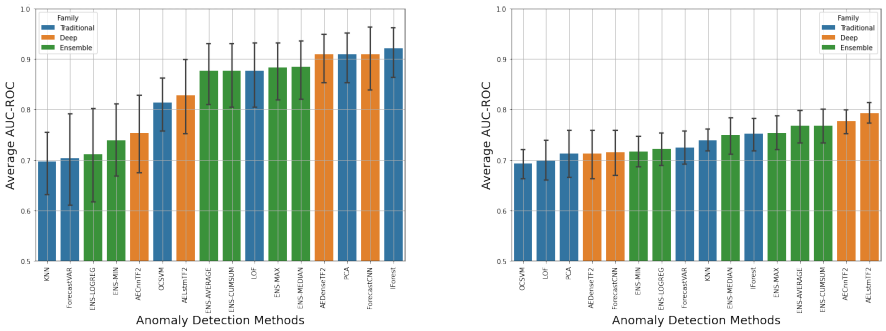
The evaluation of the anomaly detection system has involved two servers' hostgroups, one for the "Batch" use case and the other for the "Shared" use case. Two datasets covering 11 time series metrics of about 200 servers in 6 months of operation have been analysed and annotated using the procedure described in Sec. 4.1. The performance metrics selected are: system load, CPU percentage (idle, system and user), amount of free memory, amount of free and swap space, I/O memory, and I/O network traffic (Fig. 1). The amount of periods flagged as anomalous were found to be 20% and 4% for the "Batch" and "Shared" datasets respectively.

A number of anomaly detection algorithms has been integrated in the system, ranging from traditional methods to deep learning ones. The evaluated traditional methods are One Class SVM (OCSVM) [11], Local Outlier Factor (LOF) [12], Isolation Forest (IFOR) [13], Principal Component Analysis (PCA) [14], K-Nearest Neighbors (KNN) [15]. The interest in the traditional methods is their relative simplicity and lower computational cost, beneficial in view of a large anomaly detection deployment. The evaluated deep learning methods are

[2]https://github.com/grafana/grafana/issues/24674

AutoEncoder Fully Connected (AEFC) [16], AutoEncoder with CNN (AECNN) [17], AutoEncoder with LSTM (AELSTM) [18] and Forecaster based on CNN (FORCNN) [19] In addition few ensemble strategies have been added: the Max and Min of scores, the Average of Maxima, the Simple Average of scores, the Cumulative Sum and the Convex Linear Combination of scores [20]. Most of the algorithms have been imported via the *PyOD* library [21].

These models and algorithms have been compared using the default parameters proposed by the adopted libraries, leaving the optimization task for a future work. The algorithm performance has been evaluated separately for "Batch" and "Shared" datasets using the Area Under the ROC Curve (AUC-ROC) [22] as figure of merit. For each dataset the analysis has been performed independently on consecutive weeks, and the weekly based AUC-ROC value has been measured using the available dataset annotations. The average and standard deviation of these multiple measurements is reported in Figure 8, where traditional, deep learning and ensemble methods are reported [3].



(a) "Batch" dataset (b) "Shared" dataset

Figure 8. Average (central values) and standard deviation (error bars) of the weekly AUC-ROCs' values measured on the "Batch" (a) and "Shared" (b) annotated dataset, for all the studied algorithms (x-axis): traditional (blue) , deep learning (orange) and ensemble (green) methods.

Far from being conclusive, this first study confirms how the different patterns of usage of the "Batch" and "Shared" servers results in different performance of algorithms examined. Further studies are needed in order to delve into the hyper-parameters of each algorithm and identify the best working point for each of them.

6 Conclusions

The detection of anomalies in data centre's metrics using machine learning algorithms is an active field of novel development, with promising advantages in terms of increased quality of service and cost efficiency. At the same time, the most important players in the field, namely large commercial data centres, often prefer to keep their research findings undisclosed to increase their competitive advantage. This work attempts to fill the gap, providing a modular anomaly detection system, developed with open-source services, that can be easily adopted and/or adapted by other data centres and service managers.

Besides the data engineering task, this work embraces also the data science task via a mathematical formulation of the anomaly detection problem for time series and the implementation of multiple algorithms, in both traditional and deep learning fields. In addition, this

[3]These studies have been the subject of a master thesis [23], where more details can be found.

work highlights the need of field-specific annotated datasets and proposes a straightforward solution based on *Grafana* annotations to collect them. To prove that the approach is viable, the implemented algorithms have analysed two real datasets related to a subset of the CERN's servers and have produced their anomaly predictions. The performance of the algorithms has been measured consequently.

Future work will focus on building a larger annotated dataset, to identify and optimise the most performing algorithms for the daily detection of anomalies in the CERN cloud infrastructure.

References

[1] B. Moreira, S. Trigazis, T. Tsioutsias, EPJ Web Conf. **214**, 07031 (2019)
[2] *Openstack*, https://www.openstack.org/, accessed: 2021-02-16
[3] *CERN OpenStack infrastructure overview*, https://monit-grafana.cern.ch/d/000000024/cern-openstack-overview?orgId=3, accessed: 2021-02-16
[4] A. Aimar, A. Aguado Corman, P. Andrade, J. Delgado Fernandez, B. Garrido Bear, E. Karavakis, D. Marek Kulikowski, L. Magnoni, EPJ Web Conf. **214**, 08031 (2019)
[5] *Airflow operators*, https://airflow.apache.org/docs/stable/_api/airflow/operators/index.html, accessed: 2021-02-16
[6] *CERN Cloud Infrastructure Anomaly Detection - code repository*, https://gitlab.cern.ch/cloud-infrastructure/data-analytics, accessed: 2021-02-16
[7] S. Shen, V. Van Beek, A. Iosup, pp. 465–474 (2015)
[8] *Grafana annotations*, https://grafana.com/docs/grafana/latest/dashboards/annotations, accessed: 2021-02-16
[9] *Grafana annotation extension*, https://gitlab.cern.ch/cloud-infrastructure/data-analytics/-/tree/master/grafana_patch, accessed: 2021-02-16
[10] D. Thain, T. Tannenbaum, M. Livny, Concurrency - Practice and Experience **17**, 323 (2005)
[11] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor, J. Platt, NIPS **12**, 582 (1999)
[12] M.M. Breunig, H.P. Kriegel, R.T. Ng, J. Sander, SIGMOD Rec. **29**, 93–104 (2000)
[13] F.T. Liu, K.M. Ting, Z. Zhou, Eighth IEEE International Conference on Data Mining pp. 413–422 (2008)
[14] M.L. Shyu, S.C. Chen, K. Sarinnapakorn, L. Chang, Proceedings of International Conference on Data Mining (2003)
[15] S. Ramaswamy, R. Rastogi, K. Shim, ACM SIGMOD Record **29**, 427 (2000)
[16] C. Zhang, D. Song, Y. Chen, X. Feng, C. Lumezanu, W. Cheng, J. Ni, B. Zong, H. Chen, N.V. Chawla, CoRR **1811.08055** (2018)
[17] X. Fu, H. Luo, S. Zhong, L. Lin, Chinese Journal of Aeronautics **32**, 296 (2019)
[18] Y. Guo, W. Liao, Q. Wang, L. Yu, T. Ji, P. Li, Proceedings of The 10th Asian Conference on Machine Learning **95**, 97 (2018)
[19] M. Munir, S. Siddiqui, A. Dengel, S. Ahmed, IEEE Access **PP**, 1 (2018)
[20] A. Renda, M. Barsacchi, A. Bechini, F. Marcelloni, Expert Systems with Applications **136**, 1 (2019)
[21] Y. Zhao, Z. Nasrullah, Z. Li, Journal of Machine Learning Research **20**, 1 (2019)
[22] K. Feng, H. Hong, K. Tang, J. Wang, SSRN Electronic Journal (2019)
[23] M. Paltenghi, *Master thesis* (2020), `https://cds.cern.ch/record/2752641`