Intelligent Monitoring of Incidents in Complex Systems (November 2022)

Tiago Alexandre Mendes Marques
Instituto Superior Técnico
tiagomendesmarques@tecnico.ulisboa.pt

***Abstract*—Business companies started using computers since the middle of the last century. Nowadays, most companies, even if not technology related, depend on the correct operation of complex computational systems. Such systems generate data from several running services, microservices, and tasks that can reveal a lot about the systems' behavior. Events or observations that deviate from the normal system behavior are considered anomalous and may indicate near-future critical incidents, such as technical bugs and glitches, system malfunctions or hardware problems. However, the amount of data that can be extracted from running machines, cannot be analyzed manually due to its dimension and complexity nor can it be done by setting simple metric thresholds. Throughout the thesis, an intelligent monitoring system will be studied and implemented in order to predict recurring incidents. The eagerly sought-after perfect system is unfortunately unattainable. Incidents occur, which is somewhat accepted in the IT industry. In contrast, what is not accepted, is a repetition of a similar incident that already took place in the past. The goal of this thesis is to study and develop a monitoring system that predicts repeated or recurrent incidents. Making use of service metric datasets, provided by real world organizations. Data is fed in real time to the intelligent monitoring system, which will attempt to learn from past incidents and generate reliable future predictions. Service engineers will then use these predictions to perform preventive maintenance, rapidly adapting and acting upon changing conditions to avoid service failures.**

***Index Terms*—Artificial Intelligence (AI); Artificial Intelligence Operations (AIOps); Development and Operations (DevOps); Event Detection; Incident Fingerprint; Incident Prediction; Information Technology (IT) Monitoring; Machine Learning; Monitoring Information Technology(IT) Operations; Pattern Recognition; Preventive Maintenance; Site Reliability Engineering;**

I. Introduction

Due to ever changing business and market requirements, digital transformation has taken over all sorts of companies. Businesses, even if not technology related, now rely on computers to either sell their products/services or internal tools that employees use. As businesses scale and demand increases, companies start adopting complex infrastructures, moving from legacy on-premises to a hybrid mix between on-prem and cloud, or moving completely to a full cloud solution.

In the late 1980s, CCTA created ITIL [1], a standardized way of managing IT operations as a service.

When incidents happen the business is impacted, whether it is directly because users cannot checkout on an e-commerce website or indirectly, such as workers not being able to perform tasks in their internal tools because the company databases are down. Reliability plays an important role in delivering the best experience to the end user, monitoring applications and services' health is crucial when users rely on them in real time. The goal is to prevent incidents by solving minor issues before they impact the business' revenue or customers' satisfaction.

Service running machines generate metadata that can reveal a lot about the systems' behavior. With such a large volume and complexity of data, typical manual monitoring simply isn't effective. Trying to set up simple metric threshold alarm rules generates too many false positives due to the dynamic nature of these infrastructures. A high number of false positives leads to alert fatigue, which desensitizes IT teams and can have disastrous consequences. A study conducted by McCafee regarding alert fatigue in IT security professionals revealed "Of the 2,542 anomalous *events*, only 23.2 are actual threats, a ratio of nearly 110:1 that reveals the potential scale of false positive alerts." [2]. McCafee's study showed 31.9% of the sample data ignored alerts due to alert fatigue and as a consequence they receive more *events* than one can manage and investigate. The survey and study regards to IT security incidents, but this can be equally applied to IT service incidents.

Gathering all metrics in one place and using AI to learn the normal behavior of a service would be the desired approach to monitoring complex dynamic infrastructures. Current tools in the market focus mostly on the direct connection between metric univariate or multivariate anomaly detection to incidents. These approaches work for some use cases, however, there are two major problems. Most of the time incidents have already happened and damage has been done. Currently available approaches do not solve the recurring incident problem, which terrifies companies. An incident in a specific service is one thing, nevertheless, a repeated incident in the upcoming days or weeks damages services' reputation. Ideally, IT service engineers would be alerted in real time when an incident is predicted based on evidence found in metric behavior. This way, they can act quickly to minimize damage or even avoid the potential incident.

A. Abbreviations and Acronyms

AD - Anomaly Detection; AI - Artificial Intelligence; AIOps - Artificial Intelligence for IT Operations; APM - Application Performance Monitoring; CCTA - Central Computer and Telecommunications Agency; DL - Deep Learning; DevOps - Development and Operations; ETL - Extract, Transform

and Load; IT - Information Technology; ITIL - Information Technology Infrastructure Library; KPI - Key Performance Indicator; LOF - Local Outlier Factor; IF - Isolation Forest; LU - Luminol Univariate; LS - Level Shift; VS - Volatility Shift; HM - Hand Made; SADTK - Seasonal ADTK; SFBP - Seasonal Facebook Prophet; ML - Machine Learning; MTTA - Mean Time To Acknowledge; MTTR - Mean Time To Repair; SaaS - Software as a Service; SLIs - System Level Indicators; SLOs - System Level Objectives; SRE - Site Reliability Engineering; CSV - Comma-separated values;

II. Related Work

AIOps is a term created by Gartner [3] and it's about using AI and ML techniques to empower software and service engineers to build, maintain and operate services. The use of these techniques improves engineering productivity, reduces human operational costs and helps to ensure high reliability of services which directly translates to customer satisfaction. [4]. Even though there are tools in the market that take advantage of this to create dashboards and collaborative spaces to help solve incidents, the damage is already done and the user felt the impact. The incident happened and only then, in a reactive manner, the event correlation was useful, and the problem with this approach is that most likely service engineers are not even going to fix the root cause of the problem, they will limit themselves to rebooting the service back on waiting for it to happen again. Anomaly detection is an area of study which has become very popular in last decade. It is an important step in time series data analysis that consists in identifying outliers in a group of data points. AD algorithms include Local Outlier Factor, Isolation Forest, Level Shift, Volatility Shift, Luminol Univariate and Seasonal Facebook Prophet. In this assignment it is proposed to use a modified version of the Fuzzy Fingerprints classification method explained in [5]. This fuzzy logic approach has been widely used in the text and documents domain, by Prof. Dr. João Paulo Carvalho [6]. Now, a modified approach for the IT area of incident prediction. This study involves creating and matching fingerprints directly from the extracted events that are outputted by the algorithms. The categorization steps for the IT use case would be:

1) Extract events using outlier/anomaly detection algorithms.
2) Provide training set (events) for each category (incidents).
3) Fine tune features, correspondent to events extracted
4) Fine tune fingerprint extraction and prediction parameters

After analyzing existing solutions available in the market, one can realize there is no direct solution for the recurrent incident problem. Existing solutions focus on alerting or predicting the first time an incident happens. This is a good thing, but since the incident has already happened, the system has been impacted. The hypothesis of this study is based on accepting that, although an incident, of some type, can happen once it will be predicted and prevented in future occurrences. Other tools also connect metric abnormal behavior directly with incidents, using simple metric predictions and alerting when these seem out of normal values. This normally leads to a high amount of false alerts, leading to alert fatigue. Unfortunately, there is no algorithm, tool or platform to test this study hypothesis as it is a problem that no one has yet been able to tackle, at least according to what is publicly available online.

III. Contextualization and Dataset Description

A. Dataset

It was established that the definition for quality metrics during the study relied on three characteristics - continuous, high frequency and long time window dataset. The intent was to conduct the work in a manner that is very close to a real time, around the clock, predictive system. The available dataset was pre-processed. This included metric parsing from logs, removal of non continuous data values and a typical ML training testing split was applied. The training/test dataset was split into two parts training (2/3) and one part testing (1/3). During the data transformation stage it was chosen not to perform metric normalization. Initial training set may not include the total range of values for all feature metrics. The metrics were formatted into a simple data structure with timestamp and respective value. An example can be found in table I.

Timestamp (sec)	Metric 1	Metric 2	...
1442412000	1.455	45.321	...
1442415600	23.111	56.256	...
1442419200	4.123	48.937	...
...	...	...	...

TABLE I
EXAMPLE OF ORGANIZED PRE-PROCESSED METRICS READY FOR THE NEXT PHASE OF *event* EXTRACTION.

B. Entities definition

An ***event*** is an arbitrary behavior that is directly or indirectly related to what the underlying machine is running. It is spotted by an algorithm. These are not necessarily abnormal behavior, they can correspond to normal metric behavior as well. An *event* is composed by a timestamp, metric and machine. A behavior is not a discrete point in time, it is metric movement during a time window. Nevertheless, it defined as the initial timestamp value at which the detected movements started. The timestamp is an integer in "Unix epoch" format, which is the number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT). The metric and machine fields are strings with the metric name and machine identification, respectively.

A ***fingerprint*** is a trace of *events* that has been proven to precede an incident. Even though it differs substantially, the idea was born from the Fuzzy Fingerprints classification method [7]. It is defined by the *event* types and their frequency count for each metric. It can be defined and visualized in a "dictionary style" as follows:

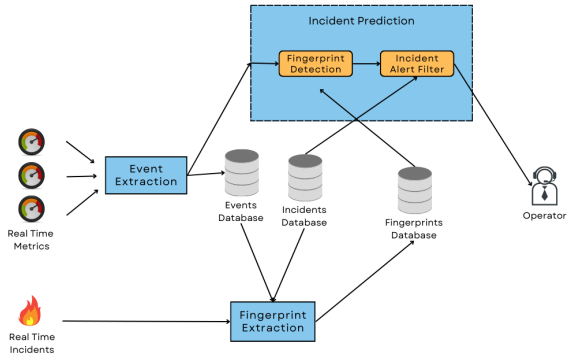

Fig. 1. Monitoring system architecture overview

```
{
    'MetricId' (string):
      {
          'EventType' (string): count (integer),
          ...
      },
    ...
}
```

The purpose of this abstract fingerprint concept is to store the events and their counts in a single object in an organized way. This fingerprint is then stored in the fingerprint database. It can later be fetched and compared its events against real time happening ones.

C. Contextualization

In figure 1 an overall overview of the thesis monitoring system modular architecture is presented. It is an ETL system pipeline approach, in which two non-synchronized branches transform data and are interconnected by a shared storage platform used to keep track of *events, incidents and fingerprints*. The top branch starts with the ingestion of real time metrics sourced from machines in which monitor services are running. Metrics start their journey at the *Event Extraction* module, where transformation extracts *events* from the "sea" of metrics, they are then stored in their respective database. At the lower branch, incidents that happen or are stored in a database are consumed by the *Fingerprint Extraction* module as well as *events* from the respective database. This module is responsible for exporting and saving a fingerprint, composed of prior *event* occurrences, that were found out to correspond to the cause of incidents in a cause-effect manner. These fingerprints are stored in a fingerprint database accessible to the top real-time branch of the system architecture. The *Incident Prediction* module is placed in series with the *Event Extraction* module. Freshly extracted *events* from metrics are loaded into the module, which can be subdivided into two serially connected sub-modules. The *Fingerprint Detection* sub-module is responsible for ingesting *events* and using a comparison engine to match *events* with the fingerprints from the database. When a match is detected it is fed to the *Incident Alert Filter*, responsible for deciding whether to alert or not the operator.

IV. Implementation

A. Event Extraction

The intention is to extract timestamp values that correspond to a certain metric behavior. It is relevant to acknowledge that these *events* do not necessarily and directly indicate abnormal or critical behavior. They represent an arbitrary behavior that is directly or indirectly related to what the underlying machine is running. To extract these behavior *events* each metric must go through the following selection of algorithms.

The more meaningful the *events* we extract, the better the data we have to later build real fingerprints that result in incidents. The following algorithms must be analyzed to understand which hyperparameters to select. These will directly influence the extracted *events*. One can already question how can this be done in a general way that would work for all machines and their running services? The selection of these hyperparameters is not trivial. Even less trivial for a generalized approach.

1) LOF [8] [9]: is used with the intent to extract anomalies by computing the local density deviation of each datapoint with respect to its neighbors [10]. In figure 2, events extracted using the LOF algorithm are marked in red dots. The *n_neighbors* parameter was chosen to be 20. With a metric resolution of 1 minute, comparing the values of the 20 neighbors surrounding the pivoting datapoint, would provide a time window long enough to spot a considerable Manhattan distance difference. If a small amount of neighbors is chosen the algorithm becomes fast, which is good, but very reactive to sudden metric changes. For this use case it would not be advantageous to get those outliers. This is strictly because sudden metric movements in small time windows are normal behavior in most IT service machine metrics. Anomalies are rare, a very small percentage of datapoints are anomalous datapoints. When choosing *contamination* this has to be taken into account. Therefore very small numbers were used and changing between order of magnitude made a tremendous difference in the number of outliers detected. After testing,

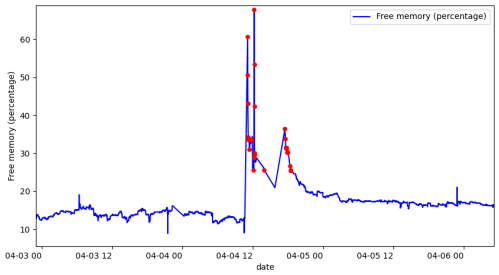

Fig. 2. Local Outlier Factor chart with marked outliers (red dots). Metric represented (in blue) is *"Free Memory (percentage)"*.

the final value selected was 0.002. When using this value, for several different metrics, the charts with marked anomalies presented what seemed an acceptable anomaly per normal datapoint ratio and good anomaly placement.

2) Isolation Forest [11]: this algorithm performs an ensemble of decision trees, it randomly splits the data space and identifies anomalies when the trees are shallow. The trees are built using the distance between datapoints. In order to extract relevant outlier datapoints we need to capture datapoints that are far from the normal values the machine operates. The contamination hyperparameter must be selected with a value that is high enough to capture the outliers, but low enough not to select many normal datapoints. The first analysis is performed using a histogram with the scores[11], calculated using formula 1. Variable *h(x)* corresponds to the average search height from the isolation trees built for feature *x*. Variable *c(n)* is the average search depth to find a general node in the built isolation trees. The total count of leaf nodes, residing at the ends of the three, is the *n*.

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{1}$$

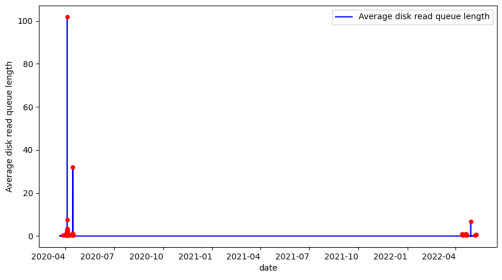

Fig. 3. Isolation Forest extracted *events* from *"Average disk read queue length"* metric.

In a specific case of metric represented in figure 3, an interesting *event* was caught where the queue jumped from very low values close to 0 to a value more than 100. This may not represent a problem as the queue may get consumed, but it certainly indicates suspicious behavior. These are exactly the types of *events* we intend to extract.

3) Level Shift [12]: it detects shifts of values by keeping a record of the median values of two sliding time windows one after the other. One of the advantages of this algorithm is that it is not sensitive to global outliers or noisy datapoints. The ambition of using this algorithm is to diagnose sudden movements in metrics that are known to be stable, in time-window defined groups or steps, along the timeline. These groups present a very low standard deviation value. They behave like plateaus, for example in figure 4.

4) Volatility Shift [12]: it detects shifts of values by keeping a record of the standard deviation values of two sliding time windows one after the other. This detector is used in parallel to the LS. While LS tracks level movements with median values, VS tracks volatility movements using standard deviation values.

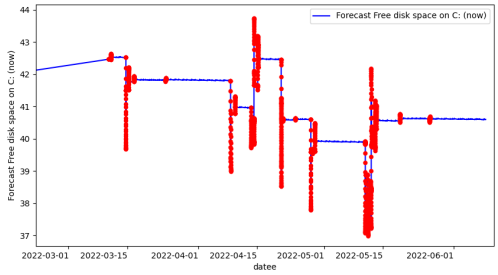

Fig. 4. Level Shift extracted *events* from *"Forecast Free disk space on C"* metric. It shows a clear plateau metric type. In red the detection of steps is presented.

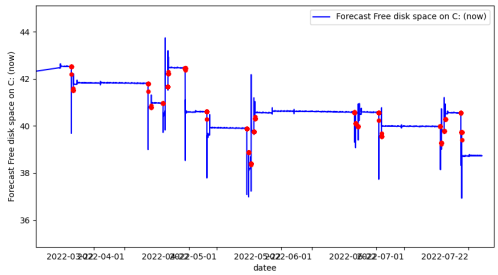

Fig. 5. Volatility Shift extracted *events* from *"Forecast Free disk space on C"* metric. Algorithm parameters were tweaked to increase sliding time windows size.

The same metric used in figure 4 was also trialed with this algorithm. Default VS parameters were tweaked to increase the sliding time window size to 60 datapoints and results are shown in figure 5.

5) Luminol Univariate [13]: is an open source Python library [13] with an anomaly/outlier detection algorithm for time series data. It was created by LinkedIn [14] to identify anomalies in real user monitoring of their applications. LU ingests time series data and calculates anomaly scores for each data point. A high score means a high probability of it corresponding to an anomaly in comparison with the other data points. In figure 6, metric "Forecast Available Memory

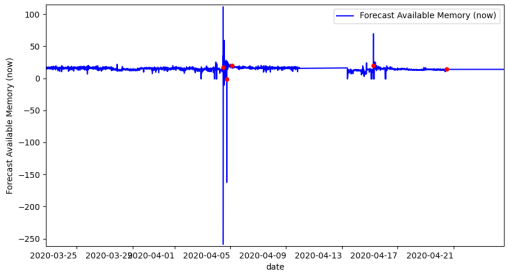

Fig. 6. Luminol extracted *events* from *"Forecast Available Memory"* metric.

(now)" shows clearly anomalous upwards spikes as well as downwards, red dots are the outlier detection performed by LU. If this metric would have been analyzed by a human and manual outlier detection was performed, the anomalous datapoints would match with the ones extracted by the LU algorithm. Of course this may not be the case for all metrics, hence the reason why these *events* are not being used directly to predict incidents.

6) Seasonal Prophet [15]: A particularity of the Prophet algorithm is the ability to point out not only global outliers but also contextual outliers. Prophet calculates what they call *trend and seasonality components*, it can be seen in figure 7. These charts represent the influence of hour, day and week on the datapoints values.

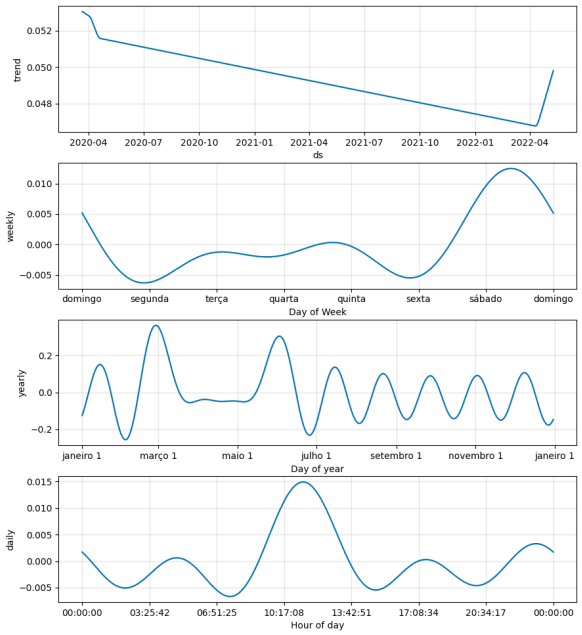

Fig. 7. SFBP trend components of *"Average disk read queue length"* metric.

Analyzing it we can disregard yearly seasonality due to the dataset available only consisting of 2 months of data. Weekly trending is quite high on Saturday compared to the other days of the week, which indicates a consistent Saturday value for this specific metric. This means, if an outlier is detected here, it certainly will be higher. Daily seasonality is interesting, as higher trends appear to be from 9 am to 2 pm. Speculating, one could find a correlation with the working hours.

7) Static Rules: Another separate attempt to get meaningful *events* was performed using static rules. These static rules were put in place specifically for each metric available. Manual task of analyzing metrics one by one, and setting out to understand which rules can be applied with the goal in mind. These static rules involved metrics above, below or equal to certain values. Another reason to use static rules is to, in the final results, study how these perform compared to the other unsupervised anomaly detection algorithms. This is a merge of the legacy static rules (already defined by most companies) with the unsupervised agnostic algorithms.

B. Fingerprint Extraction

The approach used is based on the fuzzy fingerprints classification method. In order to understand how it is applied to the use case, a simplified explanation is presented. Observe the two temporal time windows of distinct points in time represented in figure 8. Both of them resulted in the same specific incident type. In figure 9 it is represented the *event* pattern that resulted in the incident, one can observe the darker circles in figure 8 to understand the pattern of figure 9 is also there. In order to capture the fingerprint, the frequency of occurrence for each *event* type is calculated and compared against a database of other incident fingerprints.

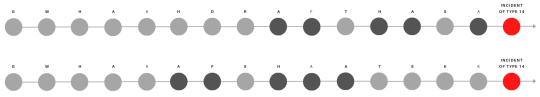

Fig. 8. Two example timelines of *events* that resulted in the same incident type. The color of the dots is related to the frequency of events. Darker dots mean more events of that type.

Fig. 9. *Event* pattern detected in figure 8.

The fingerprint extraction module required setting certain parameters. They were defined as follows:

- FINGERPRINT_INTERVAL_MINS - Δ - Time in minutes prior to an incident that *events* should be accounted for when building an incident fingerprint. This defines the time window of a fingerprint. If, for instance, a 30 minute time window were to be selected, during the learning process when an incident happens, all *events* in the last half hour are held responsible.
- FINGERPRINT_SIMILARITY_MARGIN - α - Margin of acceptance between a fingerprint and a potential fingerprint. When a new fingerprint is encountered it is compared to other already defined fingerprints. This margin is the accepted similarity difference *event* type. A direct example is a potential fingerprint that contains 3 *events* of type IF and an already defined fingerprint containing 4 *events* IF, (the difference is 1 event). If this parameter is set to one or more, then this potential fingerprint will be acknowledged and that fingerprint count is incremented. It is only acknowledged if it meets the requirement for all *event* types of the fingerprint. The fingerprint count is used in order to decide which fingerprints will later be used. This is explained in more detail below in MINIMUM_FINGERPRINTS_FOR_STORAGE.
- MINIMUM_FINGERPRINTS_FOR_STORAGE - β - Minimum number of fingerprints counted for use in incident prediction phase. This parameter regulates which fingerprints should be used in the later stages based on how many similar fingerprints were found. The reasoning behind this is that a fingerprint that only happened once is more likely to be a false-positive compared to the ones that took place numerous times. Fingerprints that present

a count value lower than this parameter will not be used in the incident prediction stages.

- N_OF_TYPES_PER_FINGERPRINT - γ - Minimum number of *event* types in a fingerprint. This parameter serves to increase*event*type diversity in fingerprints. A fingerprint that is constituted by more than one *event* type is, in principle, more trustworthy. The reasoning behind this statement is that *events* from different types can be directly translated into distinctive metric behaviors. The hypothesis behind this study, is that in the prior moments of an incident odd metric behavior can be detected using different anomaly/outlier time series detection algorithms. Hence the relevance in enforcing a fingerprint to behold more than one *event* type, since it will represent more than one odd behavior, which increases incident fingerprint confidence.

When developing and implementing the algorithm, initial parameters were required to be set. They influence directly the time window size and sensitivity of the fingerprint detection module. These variables include the fingerprint interval, number of types per fingerprint to be accepted, similarity margin and minimum fingerprints for storage described above. These variables were trialed with a span of values considered adequate to the use case.

C. Incident Prediction

This is the last component of the monitoring system. It is the one that intends to deliver value, in the form of correct incident predictions, to the service engineers responsible for the reliability of the service. It is responsible for the decision of alerting preemptively, giving time to the service engineers to act while there is still time before a major incident occurs again. It is the last layer between the monitoring system and the operator, and beholds a responsibility sense, maintaining a trusting relationship between man and machine. This module must fulfill the operator alerting needs, especially when intervention is needed. Minimizing false alerts is essential.

There are two inputs, the real time *events* that are detected by the *Event Extraction* module and access to the fingerprints database. It is responsible for ingesting the real time *events*, assessing the occurrences of each type in a certain time interval, time window Δ, before the current moment, and finally matching the fingerprint found with the ones in the database accounting a certain degree of error, α. The time interval is characterized as a time window that is calculated as delineated in equation 2. This time interval, as well as the fingerprint similarity margin, are defined as parameters to the monitoring system. These variables were trialed with a span of values that were thought to fit the use case.

$$\textit{Fingerprint Detection Interval (t0)} = [t0 - \Delta; t0] \quad (2)$$

The *Incident Alert Filter* module is another step in the incident detection group responsible for determining when the service engineer should be alerted. It receives the timestamp and information regarding incident fingerprint detection from the *Fingerprint Detection* module and using its built-in memory determines if an alert has already been released. If in the last $\Delta/2$ minutes (half of the fingerprint interval defined time) a prediction is made with the same fingerprint found, then it is discarded. The goal is to alert once, provide relevant metrics and let the operators handle the situation. Although alerts from the same service are not set off repeatedly, alerts from other services are set off normally and the operator is prompted immediately. Nevertheless, if different services run on the same machine if one fails, the other one is also likely to do so.

V. Results and Discussion

The monitoring system developed during this study was tested using different combinations of parameters Δ, α, β, γ and $algos$. The decision was to use **precision**, **recall**, **f1 score**, **f0.5 measure** and **f2 measure** as the monitoring system performance indicators, while tuning the parameters and event extractor underlying hyperparameters.

Excerpts of the best results can be found in tables II, IV and VI. Note that these trials have the *Incident Alert Filter* module turned off. It is responsible for filtering the repetition of predictions of the same incident. This was turned off so we can first analyze the *Fingerprint Detection* module predictions without the filtering module. The table VIII, shows an excerpt of the best results with the *Incident Alert Filter* module turned on.

TABLE II
EXCERPT OF TRIAL RESULTS FOR FIXED NUMBER OF EVENT TYPES, $\gamma = 1$. Δ ALGORITHM PARAMETER REFERRING TO THE INTERVAL IN MINUTES CONSIDERED FOR A FINGERPRINT, α ALGORITHM PARAMETER REFERRING TO THE FINGERPRINT SIMILARITY MARGIN, β ALGORITHM PARAMETER REGARDING MINIMUM NUMBER OF FINGERPRINTS

Δ	α	β	algos	**precision (%)**	**recall (%)**	**f0.5m (%)**
5	15	2	IF	82.18	2.87	12.6
5	999	2	IF	82.18	2.87	12.6
20	5	10	IF	80.82	2.87	12.58
30	999	2	VS	54.55	61.05	55.74
30	15	2	VS	54.52	60.73	55.66
30	999	10	VS	54.68	57.62	55.24
30	15	10	VS	54.79	57.22	55.26
5	5	10	LU	60	1.8	8.02
5	15	10	LU	60	1.8	8.02
5	999	10	LU	60	1.8	8.02
30	999	2	HM	59.16	51.6	57.47
30	999	2	LS	55.11	21.75	42.17
30	999	2	SFBP	52.75	13.81	33.72
30	15	2	LOF	51.68	10.42	28.83

In table II, precision ranges from 0 to 82.18%, while recall ranges from 0 to 61.05%. The goal is to create a monitoring system for recurring incidents, therefore precision is of greater importance when compared to recall. The system is not expected to predict all happening incidents. It is expected to predict the recurring ones and do it with high confidence values, translated to performance analysis nomenclature, precision. Interestingly, varying Δ, α, β parameters do not show major differences in precision, however, we can see a heavy

influence in recall. Sorting the table for the highest f1 score we can see VS algorithm dominating, as it seems to have the best balance between precision (ranging from 50% to 55%) and recall (ranging from 45% to 61%). Once examining the number of incorrect predictions, FP, one can understand the alert fatigue consequences. Hence the choice not to consider those trials as most appropriate for a production environment. Sorting the table by event type algorithm we can analyze the real influence of Δ, α, β. VS seems to maintain the results no matter the parameters. Δ does not impact the monitoring system's performance. This is a sign that the fingerprint in the moments before the incidents are proving a constant event count during different time-window intervals. As it seems, α also does not influence final values. This demonstrates a high similarity of event counts, if the event count numbers across the same type fingerprint are exactly the same, then varying α will not influence results. Finally, β, which represents the minimum number of fingerprints for storage also does not influence precision, only recall values. An explanation for this may be related to the number of fingerprints of the same type found. Arbitrary $\beta = 2$, $\beta = 10$ and $\beta = 60$ values were trialed, these may not influence precision due to the normal frequency of recurring incident fingerprints being higher. In fact after analysis, frequency counts ranged from 1 to 192. High frequency incident fingerprints are in fact the recurrent incident that we intent to predict, therefore we limit the minimum of fingerprints to a reasonable value.

Let us investigate other event types. Higher Δ seems to be correlated with higher precision, as a 30 minute time window for the fingerprint is capable of gathering more information, this indicates a 5 or 10 minute fingerprint may not be enough. No matter the event type, parameter α has no influence on results. α trials values 5, 10 and 999. The higher value, 999, is intended to represent infinite similarity, this is the case in which the fingerprint comparison is carried out using only the metrics that itself is composed of, not limited by event type count. Minimum fingerprint count, β, has a huge impact, as a higher β inhibits the system's predictions. This is interesting, especially because, after analysis, fingerprint counts range from 0 to 192, yet β, limiting fingerprints to the ones that have occurred at least 60 times prejudices the system. This is at least counter-intuitive, as one would expect the most repeated fingerprint would happen more times. This is probably a misconception, the reason this happens is most likely related to the high saturation of events leading to false fingerprints that just happened to be present before incidents and were wrongfully labeled as fingerprints that precede incidents. Reducing this value increases precision in most algorithms.

One event type conditions, $\gamma = 1$, seem to prove there is hope in correct incident predictions. Nevertheless, values range far from acceptable. Having 54.55% precision and 61.05% recall is not bad but it still translates to an alarmingly number of false-positives, 17837. Service engineers can lose confidence in a system that exports too many false-positives, hence the importance of alert fatigue throughout this study.

Precision is higher, even reaching 100%. This could be expected, as two event types lead to a higher diversity in fingerprint classification. Recall dropped to really low values, averaging around 2%, this is explained by the similarity of incidents, and now with this two-event type approach, $\gamma = 2$, incidents that used to be fitted in the same fingerprint now show distinct ones. One of these might not even reach β minimum count values, hence not being predicted. It would be expected that by increasing δ, precision would rise and recall would get lower, nevertheless, that did not happen. The reasoning behind this might have to do with low fingerprint count, which is a result of the criteria for fingerprint similarly.

It is interesting how the number of predicted incidents lowered drastically, proving a more prudent decision by the system. Combination of algorithms LU:VS, LU:IF and LS:SFBP show high precision results with almost no wrong predictions. LS combined with SFBP seems to have the best precision results, keeping in mind alert fatigue. Now, with two event types the influence of α is more noticeable, increase upwards of 100% on the number of predicted incidents is shown. Algorithms LS and SFBP, show a clear increase in prediction numbers while maintaining precision values. This phenomenon can be attributed to the similarity margin increase, the higher the error margin the higher the predicted incidents count. Since precision is maintained, opting for a lower error margin guarantees fewer predictions, reducing alert noise. Another interesting comparison worth noticing is that now, β values influence the predicted incident count. This may be due to, in comparison with one event type, incident fingerprints having more "resolution", hence higher number of distinct fingerprints are found. This means, on average, there are fewer occurrences of distinct fingerprints, as they were split into new ones. In conclusion, there are more fingerprints leading to a reorganization of occurrences between them, which were before concentrated in more "general" fingerprints.

TABLE IV
EXCERPT OF TRIAL RESULTS FOR FIXED NUMBER OF EVENT TYPES, $\gamma = 2$. Δ ALGORITHM PARAMETER REFERRING TO THE INTERVAL IN MINUTES CONSIDERED FOR A FINGERPRINT, α ALGORITHM PARAMETER REFERRING TO THE FINGERPRINT SIMILARITY MARGIN, β ALGORITHM PARAMETER REGARDING MINIMUM NUMBER OF FINGERPRINTS

Δ	α	β	algos	**precision (%)**	**recall (%)**	**f0.5m (%)**
5	5	2	LS;SFBP	100	0.4	1.96
15	5	2	LU;VS	100	0.08	0.4
5	5	2	LU;IF	100	0.08	0.4
5	15	2	LU;IF	100	0.08	0.4
5	999	2	LU;IF	100	0.08	0.4
20	5	2	VS;SFBP	100	0.04	0.2
5	5	2	LU;LS	100	0.04	0.2
15	5	2	LS;SFBP	100	0.04	0.2
20	5	2	LS;SFBP	100	0.04	0.2
30	5	10	HM;LS	100	0.04	0.2
30	5	60	HM;LS	100	0.04	0.2
30	999	2	HM;VS	55.17	27.29	45.81
30	999	2	HM;IF	60.68	15.92	38.84
30	15	10	HM;VS	54.37	16.16	36.91
30	999	2	HM;LOF	59.88	15.4	37.96
30	999	10	HM;IF	60.74	15.28	38.08
30	999	2	LS;VS	54.55	12.33	32.38
30	999	2	HM;SFBP	60.51	12.13	33.66
30	999	2	IF;VS	55.93	10.18	29.45

The problem with these general fingerprints is that they are contaminated by nonuseful events. This causes the fingerprint extraction module to mix what should be distinct fingerprints into a single one, resulting in predictions with a very high absolute number of false-positives. HM and VS combined are able to reach 27.29% recall with 55.17% precision, nonetheless 10954 false positives were found. Maybe with the *Incident Alert Filter* module turned on these number will improve.

Let us compare statistics for the trial with algorithms LS:SFBP, $\Delta = 5$, $\alpha = 5$, $\beta = 2$. In the trial with 57 predicted incidents, there was an average prediction interval time of 3 minutes and 38 seconds, meaning on average the service engineers would have a heads-up of the upcoming incident 3 minutes and 38 seconds before it happens. The median was around 3 minutes and 48 seconds, with values ranging from 2 seconds up to 7 minutes and 30 seconds.

TABLE VI
EXCERPT OF TRIAL RESULTS FOR FIXED NUMBER OF EVENT TYPES, $\gamma = 3$. Δ ALGORITHM PARAMETER REFERRING TO THE INTERVAL IN MINUTES CONSIDERED FOR A FINGERPRINT, α ALGORITHM PARAMETER REFERRING TO THE FINGERPRINT SIMILARITY MARGIN, β ALGORITHM PARAMETER REGARDING MINIMUM NUMBER OF FINGERPRINTS

Δ	α	β	algos	**prec (%)**	**recall (%)**	**f0.5m (%)**
5	15	2	HM;LU;LS	100	0.08	0.4
30	5	10	HM;IF;LS	100	0.08	0.4
20	5	2	LU;IF;LS	100	0.08	0.4
30	5	2	LU;IF;LS	100	0.08	0.4
15	5	2	LU;IF;VS	100	0.08	0.4
30	5	2	LU;LS;VS	100	0.04	0.2
15	5	2	IF;LS;SFBP	100	0.04	0.2
20	5	2	IF;LS;SFBP	100	0.04	0.2
5	15	2	HM;LU;IF	82.58	0.2	0.99
5	999	2	HM;LU;LS	78.39	0.16	0.79
30	5	2	HM;IF;LS	78	0.08	0.4
30	999	2	IF;LS;VS	55.99	8.26	25.97
30	999	2	HM;IF;LS	57.21	5.27	19.25
30	999	2	HM;LS;VS	56.4	4.71	17.65

Let us analyze results with, $\gamma = 3$, which indicates the need for three event types to define a fingerprint. The logical thinking behind it is: more event type diversification more features to extract and further identify a fingerprint. In table VI, the most interesting results can be found for $\gamma = 3$. This time there are 8 trials with 100% precision, nevertheless, recall is very low. Just like $\gamma = 2$, it is proven it is possible to precisely predict future incidents based on learning past behavior. Let us further analyze the algorithms used and how parameters influence the results.

Two parameters might catch one's attention, α and β. Isolating the 100% precision trials it is noticeable $\alpha = 5$ in most trials. Following $\alpha = 15$ in less precise trials. The lower the error similarity margin, the more precise the system gets. Next up, β trials perform in a rather interesting way. The higher the β the more selective the system should be, in the sense, it only uses that fingerprint for prediction if it happens β number of times. The logic is, if a fingerprint only happens once, then it is probably not worth predicting an incident based on that fingerprint data. Nonetheless, trial results prove precision works the opposite way. A minimum of 2 fingerprints achieve the best results, with exception of only one trial, in which $\beta = 10$. Why do low β values get the best results? This can be explained by the number of fingerprint occurrences being very low. Using three algorithms to extract three event types makes it very difficult or unlikely to catch the same fingerprint multiple times. After analysis fingerprint counts for these parameters ranges between 1 and 15, which adds up to this reasoning. Also, it is important not to forget that there is also a metric limitation in the fingerprint, as it is needed at least two metrics for a fingerprint to be recognized, hence the reason why the fingerprint count is so low. Predominant algorithms which achieved the best precision results are the combination of IF:LU:VS with 25 predictions and also the trial composed by HM:LS:LU with 22 total predictions. Let us compare the statistics of these two trials. In the trial with 25 predicted incidents, there was an average prediction interval time of 4 minutes and 46 seconds, warning the service engineers almost 5 minutes before it happens. The median was around 5 minutes and 38 seconds, with values ranging from 2 seconds up to 7 minutes and 30 seconds. The trial with 22 predictions averaged 7 minutes and 8 seconds of prediction interval time, a median of 7 minutes and 7 seconds, with values ranging from 6 minutes and 48 seconds to 7 minutes and 30 seconds. Even though this trial predicted three incidents less it performs best in terms of average and median prediction interval time values, but also in the worst case scenario the service engineer would have 6 minutes and 48 seconds to perform preventive maintenance. That is more than enough for the service engineer to be prepared to act, MTTA is zero, hence minimizing MTTR, which can be translated directly into downtime [16].

Running trials for different parameters shows the best performance of precision and correctness results with $\gamma = 2$, which presented double the number of correct predictions. Nevertheless, one should not discard $\gamma = 3$ trial runs. They showed almost double the interval prediction time, this increases the chances of the incident not occurring since service engineers have more time to perform preventive maintenance work once they get alerted. Why should the monitoring system be restricted by the use of just one parameter combination? This will be further discussed in the Conclusions chapter.

In table VIII, the *Incident Alert Filter* module was turned on. Comparing these results with ones in previous tables II, IV and VI it is noticeable the reduction in incident predictions. Repeated predictions accounted for the precision discrepancy between the previous trials and theses ones. The consecutive repetition of predictions spammed the timeline with true-positives and false-positives around the real incident. This spam of predictions may also be responsible for the drop in recall in table VIII, since spam predictions would accidentally match other real incidents. Comparing trial $\Delta = 30$, $\alpha = 15$, $\beta = 2$, $\gamma = 1$, $algo = VS$ of table II against the same trial specification parameters in table VIII there is a precision increase and a slight recall decrease. It was also noted that the number of predictions lowered drastically, from 17707 to 2412, which is really close to the number of real incidents.

TABLE VIII
EXCERPT OF TRIAL RESULTS WITH *Incident Alert Filter* MODULE TURNED ON. Δ ALGORITHM PARAMETER REFERRING TO THE INTERVAL IN MINUTES CONSIDERED FOR A FINGERPRINT, α ALGORITHM PARAMETER REFERRING TO THE FINGERPRINT SIMILARITY MARGIN, β ALGORITHM PARAMETER REGARDING MINIMUM NUMBER OF FINGERPRINTS

Δ	α	β	algos	**prec (%)**	**recall (%)**	**f0.5m (%)**
30	15	2	VS	97.35	53.71	83.74
30	15	10	VS	97.71	49.52	81.79
30	999	2	VS	97.19	41.98	76.95
30	999	10	VS	97.39	38.71	74.73
20	15	2	VS	74.57	37.35	62.18
20	15	10	VS	75.02	33.96	60.41
20	999	2	VS	72.98	32	58.1
20	999	10	VS	73.71	29.13	56.44
15	15	2	VS	55.68	27.97	46.47
15	999	2	VS	55.06	25.74	44.84
30	5	2	LS	93.08	21.31	55.62
15	15	10	VS	56.22	24.74	44.82
15	999	10	VS	55.68	22.91	43.29
30	15	2	LS	92.84	19.55	53.06
30	15	10	LS	92.67	18.6	51.58
20	5	2	LS	70.37	18.2	44.72
30	999	2	HM;VS	96.37	15	46.23
30	15	2	HM;VS	96.79	14.45	45.23
20	15	2	HM;VS	77.46	12.73	38.4
30	15	10	HM;VS	98.25	11.73	39.7
15	5	2	HM;VS	53.29	12.17	31.8
20	999	2	HM;VS	75.06	10.89	34.46
30	999	10	HM;VS	96.39	10.14	35.67
30	15	2	IF;VS	96.02	8.82	32.25
30	15	2	LS;VS	94.35	8.34	30.81

In this trial, there was an average prediction interval time of 7 minutes and 11 seconds. The service engineer is warned 7 minutes and 11 seconds before it happens. The median was around 7 minutes and 5 seconds, with values ranging from 1 seconds up to 12 minutes. In trial $\Delta = 20$, $\alpha = 15$, $\beta = 2$, $\gamma = 1$, $algo = VS$, there was an average prediction interval time of 4 minutes and 28 seconds. The median was around 4 minutes and 43 seconds, with values ranging from 1 seconds up to 10 minutes.

Table VIII does not contain $\gamma = 3$ results since recall values were less than 0.1%. During planning and implementation, there was the hypothesis of achieving better results if $\gamma > 1$. Albeit results with the *Incident Alert Filter* module turned off presented higher precision values as hypothesized, when turning it on, $\gamma = 1$ showed to have better precision. Also the recall was consistently higher in $\gamma = 1$ results.

VI. CONCLUSIONS

During outlier detection algorithms implementation, hyperparameter choice is of the highest importance. Metrics and corresponding algorithms were carefully analyzed and parameters were chosen accordingly to the use case. Nevertheless, the conclusion was arbitrary values seem to show equal performance results to specific use case scenario parameters. However, for future work, one cannot discard a heuristic approach for an agnostic solution. Implementations such as [17], may impact the event extraction process, hence having relevant impact in later fingerprint detection and prediction stages.

The incident prediction module uses real time extracted events from the event extraction module and matches those with already acknowledged fingerprints. This is performed in a direct comparison way. As a further improvement, a fingerprint weighted decision is suggested. It may outperform these study results. It is accomplished by giving more relevance to certain event types, as in ones are more relevant than others. The way to implement this would be in the fingerprint extraction module. The module would perform the fingerprint extraction in a rather different and more complex way. Incident fingerprint is the collection of events occurring, in an arbitrary Δ, before an incident happens. These event collections of incidents of the same type would be compared against each other. The goal would be to discard outlier events, events that appear just rarely and establish a weighted distribution for each event type. This weighted distribution would be linear to the event type occurrences, to do so, for instance applying the division by the greatest common divisor. This way, in the incident prediction module, proportional fingerprints can now be found. It is also relevant to point out, the usefulness of discarding outlier events, specially in a machine running more than one service. Having several services running on the same machine sharing components introduces metric noise, as the metrics represent the sum or aggregation of both service processes running at the same type. Critical machine problems shouldn't be affected by this as extracted events from metrics should reveal these anyway, however service problem detection is negatively influenced. To combat this, using SRE might be the solution. This is further explained in the last paragraph of this chapter.

The performance results showed prove there can be a correlation between metric behavior and service incidents. It is difficult to point out the best results, since they may vary depending on the type of services being monitored and how the service engineers prefer to be alerted. If one decides to prioritize precision over recall, then the f0.5 measure is the best indicator. If favoring recall over precision then the f2 measure is the indicator to look at. This is a subjective case since it depends on the problem being tackled. Nevertheless, in my opinion, precision should, in most cases, be favored over recall, hence f0.5 measure is more appropriate. The reasoning behind this comes from false-positives leading to alert fatigue, therefore, the need to minimize them.

The hypothesis of the study is coherent with the results and there are useful insights that can be discussed and concluded. The number of event types for fingerprint extraction can be one, but the the metrics composing it needs to be at least two. Also fingerprint interval proved to work best with a 30 minute time window. In the performance results of this study, individual fingerprint types were tested in each run. It would be interesting to see if the results could be improved, as future work, with predictions made using fingerprints extracted from the best trials $\gamma = 1$, $\gamma = 2$ and $\gamma = 3$ runs. The reasoning behind this composition is to have dynamic fingerprint extraction module, that updates itself according to performance results, in an completely unsupervised manner. A scheduled update would recalculate performance results for different parameters and classify those runs according the

results of each run. Then accordingly decide whether or not to use those trial runs parameters for real world alerting. This is specially important in systems that change over time. This schedule should include important dates such as major update deployments and scalability changes. Preset parameters include arbitrarily chosen values, the same way proceeded in this study. In the end of chap:analysis, a question rose up. Why should the monitoring system be restricted to the use of just one parameter combination? In fact, it doesn't need to be limited to just one. Recalculating performance results and choosing several parameter combinations to be applied in the monitoring system would increase the overall performance. The way this works is by combining the best results. Each parameter combination will have a specific aptitude to catch its types of incidents, hopefully resulting in an overall higher precision and recall values.

Even though the goal is an unsupervised monitoring approach, incidents may occasionally be incorrectly predicted, false-positives. To deal with this, a simple supervised module could be implemented with a training feature. A false-positive incident is predicted, the service engineer acknowledged no incident happened in the following moments. The suggestion would be to add the possibility to manually activate a function to delete or reevaluate that fingerprint, whether it is deleted completely off the fingerprint database, or adjust α value, corresponding to the error margin allowed for a fingerprint to be matched with one another.

The idea of running a real time monitoring system using heavy algorithms, like IF and SFBP, may concern one about computational processing challenges. What is needed to run this setup? Well, it scales linearly, horizontally, with the number of machines being monitored, since they are independent in regards to metrics. IF algorithm uses heavy processing, nevertheless, it is not one of the best performing algorithms, so most likely it will not be used in the real case scenario. Nonetheless, SFBP presented good results, and it requires heavy processing if a significant amount of metrics need to be processed. In normal running conditions with event types already selected, only certain algorithms will be used, therefore even a dual-core with 4 threads available is more than enough to monitor 2 or 3 servers. The monitoring system leverages multi-threading to parallelize tasks. More specifically in the event extraction module. Algorithms IF, LOF and SFBP leverage multi-threading. During the study a 3 month batch of data needed to be processed, therefore multi-threading was also implemented in the fingerprint extraction and incident prediction module, splitting the time window of events and distributing it to the available 12 threads. This resulted in an improvement in runtime performance of almost 144x.

An incident database is a reliable and vital source of information for the deployment of a monitoring system like this. It may not always be available, therefore an input source must be provided, as future work a solution to this challenge is suggested leveraging one of the latest topics in the industry, SRE. SRE (Site Reliability Engineering) is a set of principles and practices first introduced in 2003, by Ben Sloss, a Service Reliability Engineer at Google. It "incorporates aspects of software engineering and applies them to infrastructure and operations problems" [18]. The suggestion is to implement SRE principles, more specifically the definition of SLIs and their respective SLOs. The way this would be implemented is by defining availability, latency, performance, and capacity metrics. SLIs are in the form defined in equation 3. In the formula, *Good Events* are for example number of successful HTTP requests or number of calls that completed successfully in less than 100 ms. *Bad Events* are the opposite, more information can be found in the SRE Google book [19]. These metrics are operational metrics, instead of infrastructure metrics. They paint the picture on the customer/end-user level. A combination of the monitoring system studied in this thesis with the input of SLIs defined by the service engineer, would result in a trustworthy source of information regarding incidents/problems that impact the business. This approach would correlate the infrastructure layer with the business and operational layers. Using the infrastructure metrics to predict business and operational incidents. Providing the root cause analysis of how it all happened. This is especially relevant to the service engineers that could use this information. They would not only get up to speed on the incident resolution process much faster but also understand the cause and further perform reliability work, in order to make sure it does not happen again.

$$SLI = \frac{GoodEvents}{GoodEvents + BadEvents} \tag{3}$$

References

[1] Central Computer and Telecommunications Agency. ITIL (Information Technology Infrastructure Library) V2, 2009.

[2] McAfee Cloud BU. Alert Fatigue: 31.9 https://www.mcafee.com/blogs/enterprise/cloud-security/alert-fatigue-31-9-of-it-security-professionals-ignore-alerts/.

[3] Gartner. AIOps (Artificial Intelligence for IT Operations). https://www.gartner.com/en/information-technology/glossary/aiops-artificial-intelligence-operations.

[4] Yingnong Dang, Qingwei Lin, and Peng Huang. Aiops: Real-world challenges and research innovations. 2019.

[5] Nuno Homem and Joao Paulo Carvalho. Authorship identification and author fuzzy "fingerprints". 2011.

[6] João Paulo Carvalho - Publications. https://fenix.tecnico.ulisboa.pt/homepage/ist14039/publicacoes.

[7] L. Kóczy. Vector valued fuzzy sets. 1980.

[8] Breunig, M. M., H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: identifying density-based local outliers. 2000.

[9] ScikitLearn. Outlier detection with Local Outlier Factor (LOF). https://scikit-learn.org/stable/auto$_e xamples/neighbors/plot_l of_o utlier_d etection.html$.

[10] Xu H, Zhang L, and Li P. Outlier detection algorithm based on k-nearest neighbors-local outlier factor.

[11] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest.

[12] ADTK Detectors Documenation. https://arundo-adtk.readthedocs-hosted.com/en/stable/notebooks/demo.html.

[13] Luminol. https://github.com/linkedin/luminol.

[14] Linkedin. https://www.linkedin.com/.

[15] Facebook. Prophet. https://facebook.github.io/prophet/.

[16] Atlassian. MTBF, MTTR, MTTA, and MTTF. https://www.atlassian.com/incident-management/kpis/common-metrics.

[17] Zekun Xu, Deovrat Kakde, and Arin Chaudhuri. Automatic hyperparameter tuning method for local outlier factor, with applications to anomaly detection.

[18] Google. Ben Treynor Sloss Interview. https://sre.google/in-conversation/.

[19] Heather Adkins, Betsy Beyer, Paul Blankinship, Ana Oprea, Piotr Lewandowski, and Adam Stubblefield. Building Secure Reliable Systems, 2020.