A Framework for Tunable Anomaly Detection

Md Rakibul Alam
Technical University of Munich
Munich, Germany
rakibulmd.alam@tum.de

Ilias Gerostathopoulos
Technical University of Munich
Munich, Germany
ilias.gerostathopoulos@tum.de

Christian Prehofer[1]
Technical University of Munich
Munich, Germany
christian.prehofer@tum.de

Alessandro Attanasi
PTV SISTeMA
Rome, Italy
alessandro.attanasi@ptvgroup.com

Tomas Bures
Charles University in Prague
Prague, Czech Republic
bures@d3s.mff.cuni.cz

***Abstract*—As software architecture practice relies more and more on runtime data to inform decisions in continuous experimentation and self-adaptation, it is increasingly important to consider the quality of the data used as input to the different decision-making and prediction algorithms. One issue in data-driven decisions is that real-life data coming from running systems can contain invalid or wrong values which can bias the result of data analysis. Data-driven decision-making should therefore comprise detection and handling of data anomalies as an integral part of the process. However, currently, anomaly detection is either absent in runtime decision-making approaches for continuous experimentation and self-adaptation or difficult to tailor to domain-specific needs. In this paper, we contribute by proposing a framework that simplifies the detection of data anomalies in timeseries-outputs of running systems. The framework is generic, since it can be employed in different domains, and tunable, since it uses expert user input in tailoring anomaly detection to the needs and assumptions of each domain. We evaluate the feasibility of the framework by successfully applying it to detecting anomalies in a real-life timeseries dataset from the traffic domain.**

Keywords-anomaly detection, data-driven decisions, data anomalies, experimentation, self-adaptive systems

I. INTRODUCTION

Large, complex, and dynamic software-intensive systems are pushing the boundaries of today's software architecture practice towards increased automation and data-driven decisions. In the words of Eoin Woods, "*large dynamic systems will require policy-driven automation rather than today's carefully designed step-by-step processes*" [1].

One manifestation of the above idea can already be observed through the increased presence and importance of continuous experimentation in today's web-facing companies such as Google [2], Microsoft [3], Yahoo! [4], Facebook [5], LinkedIn [6], Amazon [7] and Uber [8]. Such companies, but also companies from the embedded domain [9], [10], are routinely using A/B testing [11] and more sophisticated techniques such as multi-armed bandits [12] and black-box optimization [13] to compare features and feature variants based on the degree they increase the satisfaction of business goals and to optimize their operational systems.

Another manifestation of the same idea can be seen in the area of autonomic and self-adaptive systems, i.e. systems that can adapt to changes in their environment by adjusting their structure or behavior to continue meeting their goals [14], [15]. Cloud auto-scalers, self-driving cars and real-time traffic control systems are practical examples of systems that currently exhibit certain degree of self-adaptivity.

A common denominator in all cases of increased automation in software architecture practice is the extensive use of data. Indeed, decisions made by product managers, DevOps teams, and by the running systems themselves are based on the collected data and the insights they provide. Since real-life data are noisy, several samples of a single metric are typically measured, aggregated, summarized, and finally analyzed. To obtain statistical guarantees, statistical procedures such as t-tests [16] are often employed.

A problem with data-driven decision-making is that real-life data are often non-uniform and contain wrong or invalid values, all of which can bias the result of data analysis. Data anomalies can actually take different forms and have different sources. Telemetry loss is an anomaly that has been shown to skew the results of continuous experimentation in Microsoft [17]. The existence of sudden rises or drops or the breaking of a seasonal pattern in an observed timeseries are other kinds of anomalies. Anomalies can happen due to failures in the monitoring infrastructure, e.g. a failure in a hardware or software sensor, temporary network unavailability, rounding errors upon aggregation, etc. They can also happen due to unusual changes in the system that are correctly and accurately detected by the monitoring infrastructure. For instance, high peaks in the network traffic of a video streaming website or in the traffic in a city can be unusual deviations from the normal behavior that can be attributed to a particular calendar day.

Regardless of their forms or sources, data anomalies can slow down or harm decision making. Having outliers in datasets obtained when comparing two feature variants in an A/B experiment increases the noise (variance) of the data and demands more data to be collected to reach statistically significant results [4]. Having a series of very low CPU utilization values (e.g. due to a bug in the monitoring code) reported for some servers in a Cloud data center can trigger auto-scaler operations that result in an over-utilization of these

[1] Current affiliation: DENSO AUTOMOTIVE Deutschland GmbH

servers. Finally, having outliers in the timeseries used for training a traffic predictor used in a real-time traffic control system will reduce the accuracy of the predictions and hence the correctness of the traffic control actions.

Detecting and filtering out anomalies in timeseries data is therefore important for data-driven decisions taken both by human operators and by running self-adaptive systems. We argue that an anomaly detection component has to be an integral part of any experimentation pipeline and self-adaptation architecture in order to increase trustworthiness of the analysis.

The problem is that existing off-the-shelf anomaly detection modules and frameworks, such as tsoutliers [18] and Twitter's anomaly detection [19] R packages, are difficult to tune in order to work in concert with experimentation or self-adaptation processes. For example, Twitter's package expects users to provide the expected percentage of anomalies present in the data, which is often not known or hard to estimate [20]. Concretely, anomaly detection should consider the running experiments and their expected effect, e.g. whether they change the model of normal behavior of a timeseries. At the same time, the experimentation engine should be aware of the expected anomalies (e.g. due to external circumstances such as holidays) in the timeseries of a system metric.

In response, in this paper we provide a simple yet robust way to detect anomalies in arbitrary timeseries and connect anomaly detection to experimentation and self-adaptation. Our method works in two stages: identification of strong seasonal patterns and identification of critical anomaly thresholds. It is a semi-automated method in the sense that, although it assumes the involvement of domain experts in the learning phase, it can work in a completely automated way after that. Our method can be encapsulated in a component that informs either the operator of a system or the adaptation manager of an autonomous system about data quality issues that may have an impact on the functioning of the system and should be dealt with by applying corrective actions.

We have evaluated the feasibility of our method by applying it to identifying anomalies in a real-life data set from the traffic domain. Our results show that it is feasible to identify different categories of anomalies in the traffic data set with minimum input from a user.

The rest of the paper is structured as follows. Section II provides basic background and definitions of anomalies and their different types. Section III describes the eight phases of our anomaly detection framework, and Section IV provides an evaluation of our framework. Then, Section V explains how our framework is integrated in a self-adaptive system. Finally, Section VII compares our approach to related work and Section VII concludes.

II. PRELIMINARIES

An anomaly is an observation that departs from the expected or usual behavior. Anomaly detection is therefore the task of defining a region that represents 'normal' behavior and identifying the observations that lie outside of this region [21]. Our framework builds on both robust statistical techniques and expert user input for anomaly detection.

TABLE I. CATEGORIES OF ANOMALIES.

	Non-contextual	*Contextual*
Isolated	**Type I**: Extreme spikes, wrong values	**Type III**: Sudden or unusual spikes
Sequential	**Type II**: Series of wrong or extreme values	**Type IV**: Series of unusual values

An important observation is that anomalies can be broadly categorized according to two orthogonal dimensions:

- *Contextuality*, i.e. whether anomalies are independent of the context in which they occur (e.g. extremely high or low values or null values) or are deviations from the values expected to be observed in a specific context, e.g. in a specific hour, day, or week. Based on this, anomalies can be divided into *non-contextual* and *contextual*.
- *Consecutiveness*, i.e. whether anomalies are isolated from each other or occur in a sequence. Based on this, anomalies can be divided into *isolated* and *sequential*. Sequential anomalies form anomaly groups that have a certain length representing the number of consecutive anomalies in the group.

Table I depicts the four different categories of anomalies that can be identified based on the above dimensions.

Each of the four categories point to different potential sources for the existence of anomalies and also pose different challenges in anomaly detection and filtering. Type I and II anomalies are mostly straightforward to detect compared to Type III and IV, which need more sophisticated analysis than simple threshold-based methods. At the same time, Type I and III anomalies can be easier to deal with by simply removing the anomalies from subsequent analyses, whereas Type II and IV need more sophisticated methods of dealing with anomalies once these are detected. Finally, Type IV anomalies can indicate possible patterns of exceptional behavior that corresponds to a certain process, e.g. warmup of a JVM in performance micro-benchmarking, holiday with increased network traffic in a video-streaming website, etc. that could not be considered anomalies but rather expected measurements that conform to a special pattern.

In the following, we describe a generic anomaly detection framework for time series data that relies on both robust statistical techniques and expert user input and, therefore, can be used for identifying all four types of anomalies.

III. ANOMALY DETECTION FRAMEWORK

A. Overview

Our framework for anomaly detection of time series data consists of the following eight phases, while it relies on three types of inputs from expert users (Figure 1).

Phase 1: Identification of out-of-bounds and wrong values. In the first phase, an expert user provides the range(s) of acceptable values in the time series (Input A). For instance, a user might specify that values should be non-negative with an absolute maximum value of 1000. In that case, any measurements that either negative or above 1000, along with

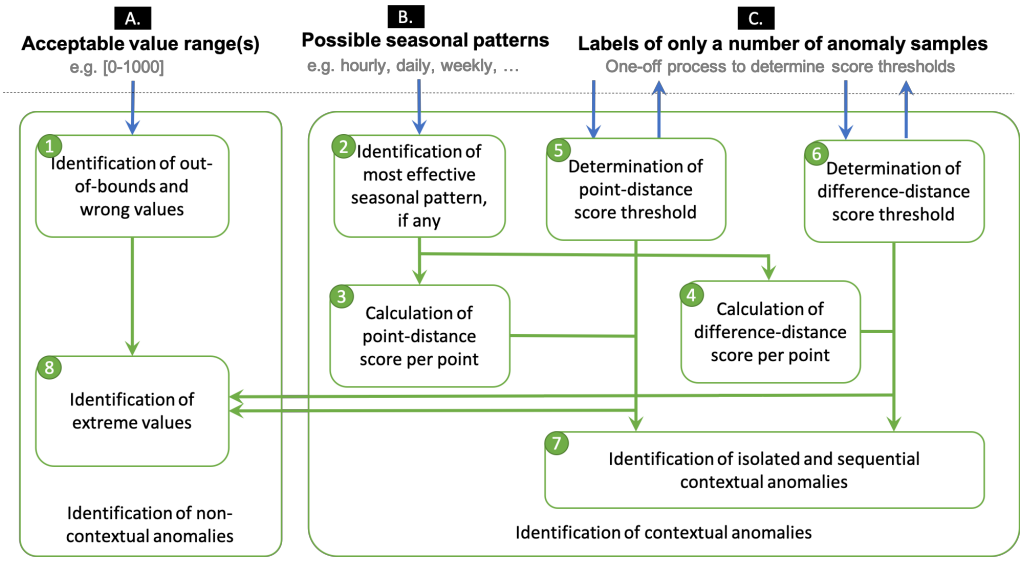

Figure 1: Overview of the proposed anomaly detection framework.

values that are not numbers or simply null are directly flagged as non-contextual anomalies in our method.

Phase 2: Identification of the most effective seasonal pattern for anomaly detection. Seasonality is an important characteristic of many timeseries and seasonal patterns can be used in obtaining a range of normal behavior from data. If there is more than one seasonal pattern, a question is how to decide which pattern is more representative of normal behavior and hence effective in anomaly detection. This phase answers this question by considering different user-provided patterns.

In particular, in this phase, an expert user provides a list of seasonal patterns that are expected to be present in the data series (Input B). For instance, a user might specify 'hourly' and 'daily' as the candidate patterns based on their experience and domain knowledge. Scatter plots of measured data can be used at this phase to inform a user's decision. Our framework then decides which pattern is the most effective for anomaly detection (if any at all). The selected pattern is used in the next phases to define the range of normal behavior. Section III.B provides a detailed technical description of phase 2.

Phase 3: Calculation of point-distance score per point. Given a seasonal pattern in a timeseries, the timeseries can be split into several partial timeseries, each of which contains data points that belong to the same similarity group. For example, a timeseries with hourly measurements that follows a daily pattern can be split into 24 partial timeseries, one for each hour in a day. One way of identifying contextual anomalies is to look for data points that significantly differ from the other points in their partial timeseries. Leveraging this idea, in this phase we focus on points that lie outside the 25th-75th percentile range of each partial timeseries dataset. For each of these points, to quantify their divergence from normal behavior, we calculate their distance from the 25th or the 75th percentile (depending on whether the point is below the 25th or above the 75th percentile), which we call *point-distance score*. Points inside the 25th-75th percentile range obtain a point-distance score of zero.

Phase 4: Calculation of difference-distance score per point. Another way of identifying anomalies is to look for points which represent sharp rises or drops. Any difference between two consecutive measurements (either rise or drop) that is unusual is a good candidate for being an anomaly. Following this, in this phase we first create the timeseries of the differences between each two consecutive points (differenced timeseries) and compute the partial differenced timeseries that capture the seasonal pattern from phase 2. Then, we calculate d_i, i.e. the distance d of each point i that lies outside the 25th-75th percentile range of the respective partial timeseries dataset from the respective boundary (75th or 25th percentile). For any point that lies within the normal (25th-75th) range it holds that $d_i = 0$. Then, each point obtains a *difference-distance score* equal to $|d_i - d_{i-1}|$, where d_{i-1} refers to the distance of the previous point in the differenced timeseries. This way, effects of sharp rises or drops on a single point are not carried over to the score of the next one.

Phases 5 & 6: Determination of point-distance and difference-distance score thresholds. After performing phases 3 and 4, each point in the timeseries is assigned two scores, a point-distance and a difference-distance score. Whereas any point with a positive score is candidate for being an anomaly, points with higher scores are more likely to be true anomalies. The question is how to set a threshold, for each of the two scores, to decide whether a potential anomaly will be actually labelled as an anomaly in our framework. This is the task of phases 5 and 6. To determine a threshold for each score, we rely on expert users providing simple (Boolean) feedback on only a subset of characteristic cases. A detailed technical explanation of these phases is provided in Section III.C.

TABLE II. BLOCKS BY DAILY INTERVAL.

	Daily Interval				
	00:00	01:00		22:00	23:00
	pts_D^1	pts_D^2	...	pts_D^{23}	pts_D^{24}
Blocks					
Day 1	***t1***	t2	...	t23	t24
Day 2	***t25***	t26	...	t47	t48
Day ...	...				
Day 20	***t457***	t458	...	t479	t480
Day 21	***t481***	t482	...	t503	t504

TABLE III. BLOCKS BY WEEKLY INTERVAL.

	Monday			...	Sunday		
	00:00	...	23:00		00:00	...	23:00
	pts_W^1	...	pts_W^{24}		pts_W^{145}	...	pts_W^{168}
Blocks							
Week 1	***t1***	...	t24		***t145***	...	t168
Week 2	***t169***	...	t192		***t313***	...	t336
Week 3	***t337***	...	t360		***t481***	...	t504

Phase 7: Identification of isolated and sequential contextual anomalies. Once the point-distance and difference-distance score thresholds have been determined, the task of identifying isolated contextual anomalies is simply to mark any data points with scores higher than the thresholds. Sequential anomalies are by definition consecutive anomalies. In our framework, we consider any anomaly that has at least one neighboring anomaly as sequential anomaly. Hence, the task of identifying sequential contextual anomalies is simply going through the list of isolated anomalies and marking as sequential the ones that have at least one neighboring point marked as isolated anomaly.

Phase 8: Identification of extreme values. In this phase, the set of non-contextual anomalies from phase 1 is enhanced by points that would be considered anomalies in all partial timeseries corresponding to the seasonal pattern. In particular, we consider anomalies as non-contextual when their (point or difference) score is either

- greater than the sum of the maximum 75th percentile value (out of all the partial timeseries) and the selected threshold, or
- less than the subtraction of the selected threshold from the minimum 25th percentile value (out of all the partial timeseries).

B. *Identification of the most effective seasonal pattern for anomaly detection*

We will describe the process our framework applies in identifying the most effective seasonal pattern in a timeseries (Phase 2 in our framework).

A timeseries ts $= \{t_1, t_2, \dots, t_n\}$ is a set of temporal data points. We define as d the time distance between each two adjacent points in *ts*. For an interval $s = m \times d$, where $m \geq 1$ and $m \in \mathbb{N}$, we can divide the whole timeseries into $c = {}^{n}/_{m}$ blocks, $b_s^1, b_s^2, \dots, b_s^c$, with m data points per block. Each point in a block is indexed by $i = j \bmod m$, where j is the point' index in ts. These blocks are equivalent to seasons in time-series literature [7]. Note that, the last block may have less than a data points, since, e.g., a month can have 28, 29, 30, or 31 days. In those cases, we pad the last block with trailing null values that do not affect the pattern comparison since they are omitted in the variance calculation performed later.

```
1.  var_ts = variance(ts)
2.  PTS = set of partial_timeseries from ts using interval j
3.  PTS_vote = 0
4.  For each ts_x in PTS:
5.    var_ts_x = variance(ts_x)
6.    if var_ts_x < var_ts then
7.      if difference is statistically significant according to F-test then
8.        PTS_vote = PTS_vote + 1
9.
10. if PTS_vote is greater length of PTS / 2 then
11.   use PTS to model normal behavior
12. else then
13.   use ts to model normal behavior r
```

Algorithm 1: Comparison of timeseries and set of partial timeseries.

```
1.  vote_TS_k = 0
2.  For each ts in TS_j:
3.    result = previous_algorithm
4.    if result is equal to 'use TS to infer' then
5.      vote_TS_k = vote_TS_k + 1
6.
7.  if vote_TS_k is greater than length of TS_k /2 then
8.    use TS_K to infer
9.  else then,
10.   use TS_j to infer
```

Algorithm 2: Comparison of different seasonal patterns.

Next, we project the t_i points of all blocks into a partial timeseries pts_s^i. By construction, pts_s^i has exactly c points. We define $PTS_s = \{pts_s^1, pts_s^2, \dots, pts_s^m\}$ as the finite set of all partial timeseries for an interval s.

Having a timeseries ts and a set of partial timeseries PTS_s, we can decide whether ts or PTS_s is more effective in representing a region of normal behavior by comparing all the partial timeseries in PTS_s with ts. We assume that a timeseries is a more effective model of normal behavior than another timeseries when it has lower variance, since then outliers will be easier to detect. If the majority of the partial timeseries are more effective than ts, we conclude that PTS_s is more effective than ts, and vice versa.

One problem when comparing variances of data sets with potentially small sample sizes (as in our case) is that a positive

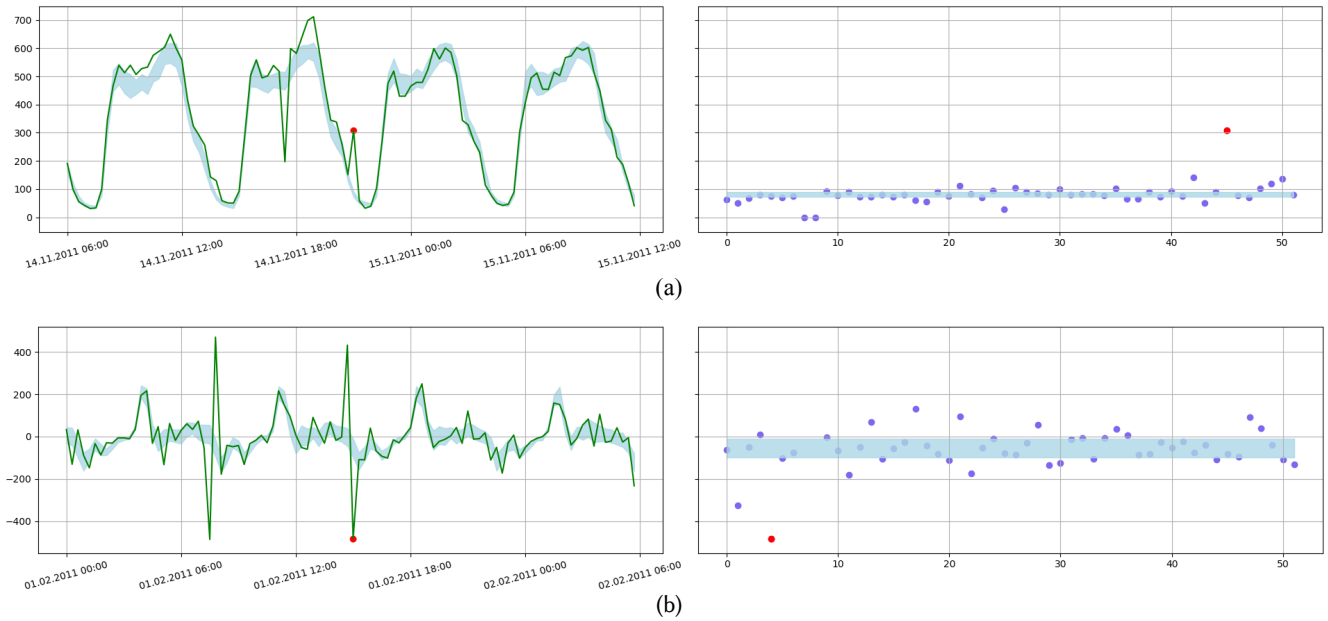

Figure 2: Diagrams shown to user for the determination of (a) point-distance score, and (b) difference-distance score. The line plot (left) shows the differenced time-series around the potential anomaly, whereas the scatter plot (right) shows the dataset corresponding to the point in the day of the potential anomaly.

or negative outcome might be a result of sampling error from the original populations. For this, we use one-tailed F-test [16] to check whether pts_s^i has statistically significant lower variance than ts. If the F-test does not report a statistically significant difference, we give a vote to ts, since at least it has more samples compared to pts_s^i and therefore can be considered more representative (Algorithm 1).

The above comparison can be used to compare an original timeseries to a set of partial timeseries generated from the original one. This technique can be iteratively applied to compare two sets of partial timeseries with each other.

Let us name the two sets of partial timeseries PTS_s and PTS_k where $l \epsilon \mathbb{N}$ and $k = l * s$. In order to deduce which of them is a more effective model of normal behavior, we can compare each partial timeseries in PTS_s with each partial timeseries in PTS_k and use a voting mechanism to sum up the results (Algorithm 2).

As an example, consider a timeseries with one-hour intervals for 21 days. For 'DAILY' and 'WEEKLY' intervals we can construct two sets of partial timeseries, PTS_D and PTS_W. PTS_D consists of 24 partial timeseries, each with 21 data points (Table II), whereas PTS_W consists of $24 * 7 = 168$ partial timeseries, each consisting of 3 data points (Table III).

To know whether 'DAILY' interval is more effective than 'WEEKLY' we start comparing pts_D^1 against pts_W^1, pts_W^{25}, pts_W^{49}, pts_W^{73}, pts_W^{97}, pts_W^{121}, pts_W^{145}. From this step we know which interval is more effective for this particular data point (e.g. Midnight, 00:00). We repeat for other data points and eventually determine which interval is more effective by applying the Algorithm 2. Note that if, for a single data point, Algorithm 1 returns e.g. 'DAILY' over 'WEEKLY', this does not necessarily mean we cannot use 'WEEKLY' pattern for anomaly detection; it only means we would not be more effective by using 'WEEKLY' in contrast with 'DAILY'.

C. *Determination of thresholds for point-distance and difference-distance scores*

We will describe the interactive threshold selection process (Phases 5 and 6 in our framework), as well as a fallback threshold selection process.

1) *Interactive threshold selection*

For each score, we perform binary search to identify score values of interest. In particular, we take the following steps:

STEP 1: Set the initial threshold as the average between the maximum and minimum value of the score.

STEP 2: Filter out potential anomalies whose score is less than or equal to the threshold.

STEP 3: Select *n* potential anomalies (we used 5) out of the remaining ones that are closest to the threshold.

STEP 4: Show a scatter plot of the time series around the anomaly point (left side of plots in Figure 2), along with a scatter plot of the partial timeseries the point belongs in (right side of plots), for each of the selected *n* potential anomalies. Mark the potential anomaly in red. Both plots show the normal behavior to help the user take an informed decision. For the point-distance score, the timeseries shows the actual values from the dataset (Figure 2(a)), whereas for the difference-distance score, the timeseries shows values from the differenced series (Figure 2(b)). The dashed regions in both plots depict the normal behavior region that corresponds to the selected threshold value.

STEP 5: Ask the user to decide whether the majority of the points marked as red are anomalies or not w.r.t. to that domain. The user answers with ‘yes’ or ‘no’.

STEP 6: Update the threshold based on response of the user by performing binary search: In case of a positive answer, the

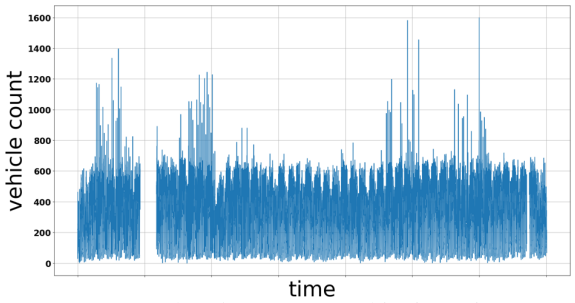

Figure 4: Raw data (15-min intervals) of LD1 for 2011.

new threshold becomes the average of the old threshold and the minimum value used in the calculation of the old threshold, so essentially anomaly detection becomes stricter. In case of a negative answer, the new threshold becomes the average of the old threshold and the maximum value used in the calculation of the old threshold.

STEP 7: Repeat steps 2 to 6 at most $\log_2 range$ times, where $range$ is the difference between the maximum and minimum of the scores of the initial dataset of potential anomalies (the input of Step 1).

STEP 8: Use the threshold value after step 7 is complete as the final threshold for anomaly detection.

2) Fallback threshold selection

In case interactive threshold selection is not an option (e.g. if zero manual effort is required or no user is available), a threshold can be selected using the modified z-score method, which is defined as follows:

$$Z_i = \frac{0.6745(x_i - \text{median})}{Median\ Absolute\ Deviation}$$

First, the modified z-score is calculated for each of the potential anomalies. Then, the point-distance or difference-distance score of the first anomaly with z-score more than 3.5 becomes the threshold for point-distance or difference-distance scores. The value of 3.5 is a recommended one for marking points as outliers [22].

D. Framework Assumptions and Limitations

In summary, the proposed framework relies on the following assumptions:

- It deals with regularly spaced, univariate timeseries. We believe this covers a rather big class of periodically monitored real-life processes, e.g. CPU load. Multivariate timeseries can be split into their univariate subparts and handled by the framework; however, this way, anomalies that result from irregularities in more than one timeseries are not identifiable.
- It can work with timeseries that have a single seasonality and no seasonality at all. In both cases, Phase 2 of the framework is skipped.
- It relies on the inter-quartile range (IQR) of data within a single block (e.g., Mondays, 11am for a year) to define normal behavior. As a result, it can work even with very small number of data points per block (which may happen if the original timeseries has collection gaps), assuming that the IQR can be still used for calculated.

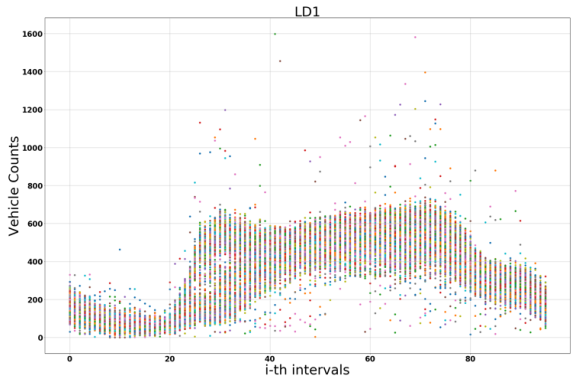

Figure 3: Daily pattern: each of the 96 points in a day corresponds to a dataset of 365 points (days in a year).

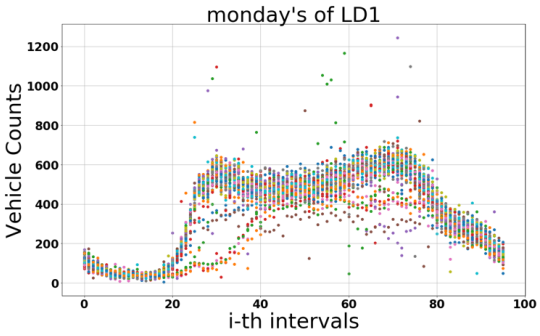

Figure 5: Weekly pattern: each of the 96 points in a Monday corresponds to a dataset of 52 points (weeks in a year). Similar for Tuesdays, Wednesdays, Thursdays, Fridays, Saturdays, Sundays.

- Data within a single block need to be normally distributed—an assumption of the F-test used in Phase 2 to compare variances.
- It classifies anomalies into sequential/isolated and contextual/non-contextual. Other well-known patterns for anomalies can be found in statistical process control, such as sequences of very high or very low values or intermittent (but not sequential) anomalies [23].

IV. EVALUATION

A. Traffic Data Set

For evaluation of the framework we have used a dataset of traffic from a road-network. These data were collected using inductive loop detectors installed on the road network of Vienna, Austria. Data from a total of 280 loop detectors (LD) were made available to us. For each LD, vehicle flow counts are recorded in every 15-minutes interval, from Jan 1st, 2011 to Dec 31st, 2014. Hence, each day yields 96 records and each year 35,040 records for each LD.

Since traffic pattern can change over time, we have used only one year's data at a time for the construction of normal behaviors. Each LD's data is used separately for anomaly detection detection. In this section, we present the results obtained from applying our framework to a single LD, *LD1*, for 2011. The results on more loop detectors, along with the source code of the developed tool are available at https://github.com/rakibulmdalam/TSADF.

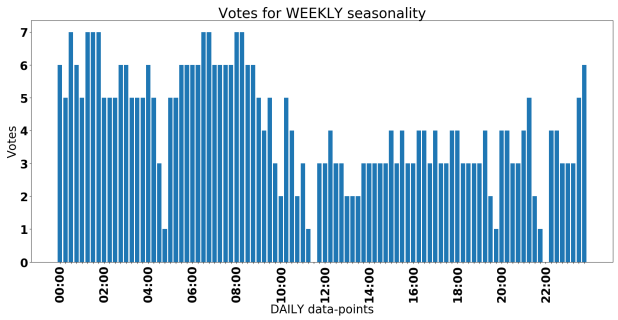

Figure 6: Votes for weekly pattern.

LD1 has significant number of high peaks which reached 1,600 VFC, where the usual peaks reach around 600 vehicles (Figure 4). Other than two visible gaps it is more or less continuous in nature which means this loop detector was able to collect data most of the time.

B. Framework Application in Detecting Traffic Data Anomalies for a Single Loop Detector

To evaluate the feasibility of our framework we have applied it in detecting anomalies present in 2011 in LD1. We report the activities and results of each phase.

Phase 1. The acceptable value range provided by the user was $[0, \infty)$. Consequently, the framework marks any points of LD1 with value outside this range as non-contextual anomalies—there are 1505 such points, all corresponding to null measurements.

Phase 2. After looking at the raw timeseries diagrams of LD1 timeseries, the user specifies the daily and weekly patterns as the most characteristic seasonal patterns present in the data. Our framework then performs all the possible comparisons between the daily pattern (Figure 3) and the weekly pattern (data from all days versus data from only Mondays, only Tuesdays, etc., see Figure 5) for each of the 96 points in a day. If the weekly pattern shows a statistically significantly smaller variance that the daily pattern in a comparison, it obtains a positive vote. Figure 6 depicts all such votes for all 96 points in a day. To make a final decision on which pattern to select for the subsequent phases, the framework counts the points in a day in which there are more than three votes (so the majority out of a total of seven votes). In these points, the weekly pattern is considered more effective. The overall result is that the weekly pattern is more than half of the cases, and as a result is selected as the most effective pattern for the next phases. Interestingly, out of the 41.2% of the cases where the weekly pattern was not statistically significantly better, in 25.4% of the cases its variance was still smaller than the variance of the daily pattern.

Phases 3 & 4. In this phase, the framework calculates the point-distance and difference-distance scores for all the points in the timeseries. The majority of scores were lower than 100 while only a small number were higher than 400. The question is, at which point to set the cutoff threshold for reporting anomaly? This is being handled in the next two phases.

TABLE IV. THRESHOLDS AND CORRESPONDING NUMBER OF THREHOLDS FOR "YES" ANSWERS OF USERS.

High tolerance user mode

	Threshold	Anomalies
Point distance score	543	16
	272	248
	204	**400**
Difference distance score	485	29
	243	154
	183	**224**

Low tolerance user mode

	Threshold	Anomalies
Point distance score	543	16
	272	248
	136	773
	68	2006
	51	3030
	47	**3372**
Difference distance score	485	29
	243	154
	122	315
	61	947
	31	**2809**

Phases 5 & 6. To illustrate these phases and in particular the impact of user input in the final result, we applied the framework with two users, one taking a low-tolerance approach where almost everything that is outside of the normal behavior is reported as anomaly, and another taking a high-tolerance approach in which only big deviations from normal behavior are considered true anomalies. Table IV depicts the decision-making of the two approaches, in particular the thresholds and the corresponding number of anomalies when the users answered positively in the question of whether the points with scores more than the thresholds are to be reported as anomalies.

Phases 7 & 8. We report here the results of following both a high-tolerance and a low-tolerance approach in labeling anomalies, as well as following the automated labeling using the modified z-score approach (Section III.C.2). The results are depicted in Figure 8. As expected, the number of reported anomalies greatly varies based on the score thresholds, and the modified z-score yields results comparable to a user with high tolerance in anomalies. With respect to the two distance scores, they are complementary in identifying anomalies, since, despite the overlapping cases where any of the two could be used (Figure 8), there are also a high number of cases where only one score is used. Finally, Figure 8 depicts also the important proportion of null values identified by the simple range filter. For illustration, Figure 7 depicts the same subset of the original timeseries annotated with the identified anomalies in the three approaches.

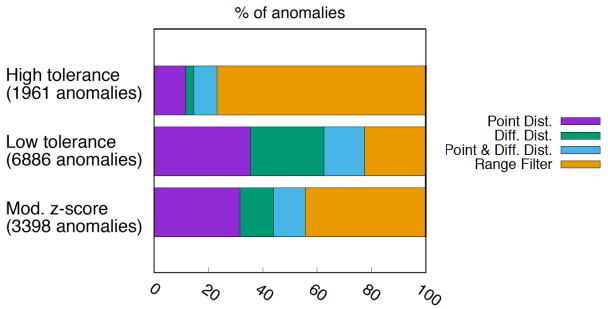

Figure 8: Sources of anomalies.

With respect to the different categories of anomalies, Figure 9 provides an overview of the different anomalies per category in the three approaches. As can be seen, the majority of anomalies in both cases are sequential anomalies; in the low-tolerance mode more contextual anomalies are reported, whereas in high tolerance mode the majority are non-contextual ones.

C. *Evaluation against manually labelled anomalies*

To obtain a baseline, one of the authors, an expert in data analysis of traffic data, performed manual labeling of anomalies in the same dataset that we used to demonstrate the application of our method, i.e. for data from LD1 in 2011. The expert worked with their tools of preference independently and without having prior knowledge of how the anomaly detection framework was used and configured to detect anomalies in the same dataset. The manual labeling process lasted about 90 mins and identified 2122 anomalies.

We compared the results from the three approaches to this baseline and calculated the accuracy, precision, recall and F1 score of our framework in each case (Figure 10). The results clearly show that the high tolerance approach performs better with accuracy of 98.5% and F1 score of 87.2%. At the same time, the labelling process with our framework, performed by two of the authors that are not experts in analysis of traffic data, took approx. 5 mins only.

Having a closer look at the wrongly annotated points by our framework, in particular at the points that were labelled as anomalies by the framework but not by the expert user, the majority of the points were within days with unusual traffic patterns, i.e. holidays. We hypothesized that holidays can be automatically detected by looking at long sequences of sequential contextual anomalies and when they happen and formulated a rule that specified that "in the high tolerance mode, a day is a holiday if it contains a sequence of contextual anomalies with length greater than 5". With the above rule, we were able to identify 11 out of a total of 13 holidays, together with 3 false positives. After removing holidays from the evaluation, all four metrics reported in Figure 10 increased.

From the above we can conclude that:

- Our framework allowed a non-expert in the analysis of traffic data to annotate anomalies about 20 times faster than an expert and with almost the same accuracy.
- Our framework scales with the number of timeseries that need to be analyzed since it only needs a small upfront input from the user and automates the rest of the analysis. As pointed out by the traffic data expert in our team, automation is absolutely necessary in real settings where manual labeling is not an option.

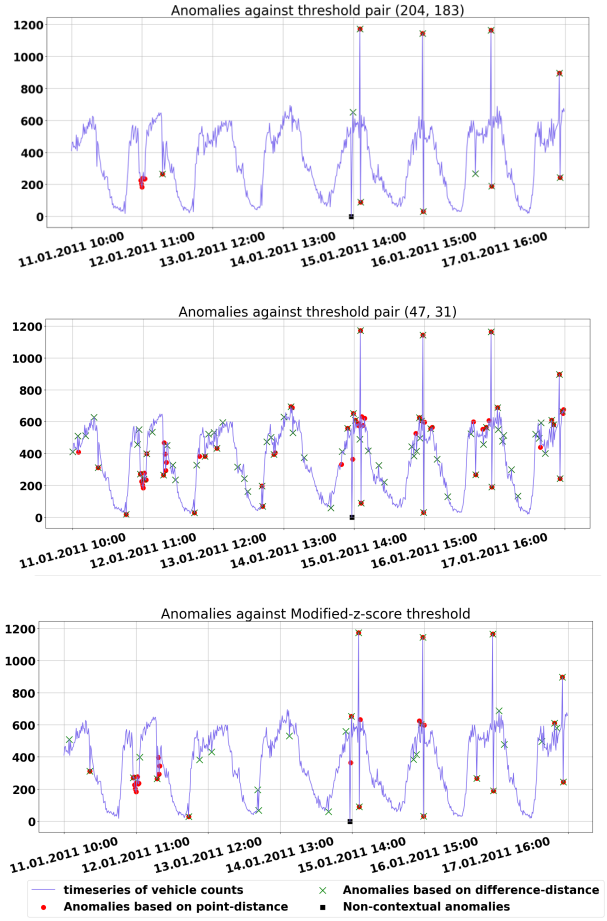

Figure 7: Anomalies identified in the high-tolerance, low-tolerance, and mod. z-score approach.

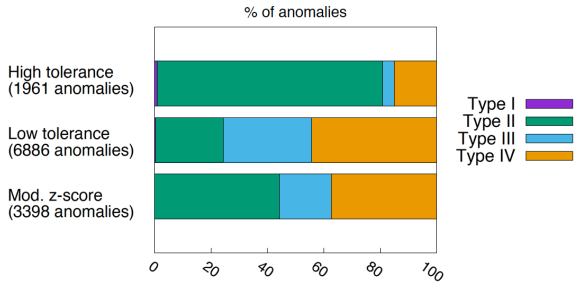

Figure 9: Distribution of anomalies in the categories of Table I.

V. Operation in Self-Adaptive Systems

Our anomaly detection framework can operate as part of a self-adaptive system in order to identify and filter out anomalous measurements at runtime. A self-adaptive typically consists of a managed subsystem and a managing subsystem, which operates based on the well-known Monitor-Analyze-Plan-Execute loop (MAPE-K) [15], monitors the

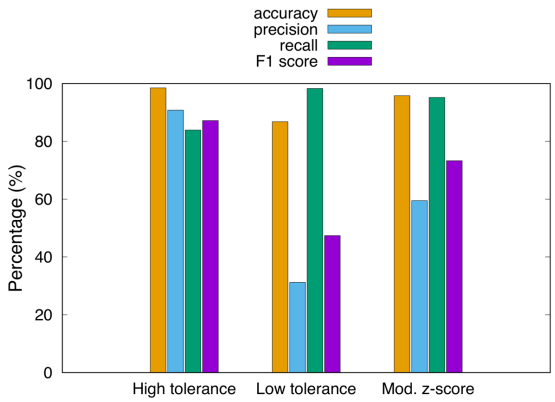

Figure 10: Evaluation of the three different approaches supported by our framework w.r.t. to the manually labelled baseline.

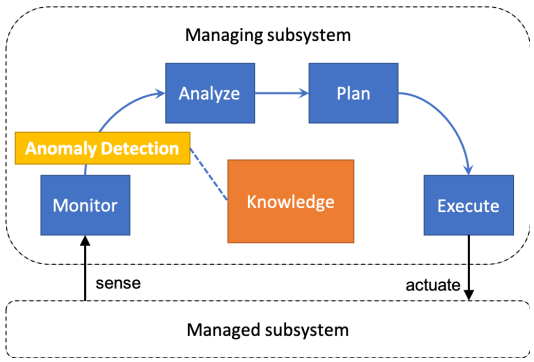

Figure 11: Anomaly detection within the MAPE-K loop.

managed system via sensors and adapts it via actuators, as depicted in Figure 11. For instance, in real-time traffic control, the managed system is the traffic infrastructure and the managing system in the traffic control system which monitors the number and speed of cars, analyzes these data to predict how traffic will develop in the near future, plans actions to avoid possible traffic congestions, and executes the plans via e.g. changing speed limits.

Our anomaly detection framework can be used between the monitor and analyze phases, or as part of the monitor phase itself. By identifying and filtering out anomalies, decision-making in the subsequent phases can be improved. Since our framework needs just initial input from the user to identify the effective seasonal pattern and select the thresholds for anomaly detection, it can work in an automated fashion until further input is needed from the user. An example is the identification of an excess of anomalies of a particular type that might point to specific problems within the monitoring infrastructure.

Our framework can also use the expert input to learn and calibrate anomaly detection. In our experiments, we used the (user-observed) association between contextual sequential anomalies with high sequence number and holidays to identify holidays and stop reporting anomalies in these exceptional circumstances. Stop reporting anomalies in holidays is just one instance of a rule that tunes anomaly detection. The framework can be tuned, via similar rules, to treat special cases differently: holidays, e.g. could be viewed as another pattern in traffic data for which a normal behavior needs to be identified and thresholds need to be learned.

We note, also, that our framework can be tuned to report anomalies at different levels of severity: for instance, after a "risky" adaptation takes place (e.g., dynamically enforcing too low speed limit), the detection module should notify the users or the automated decision-making module to revert to a safe system state based on an excess of anomalies of a particular type (e.g. non-contextual ones) pointing to a particular problem. In reverse, an adaptation that is expected to change the normal behavior for a metric should notify the anomaly detection module to not report anomalies on this metric until enough data for learning the new normal behavior are gathered. A detailed, statistical analysis of considering and balancing anomalies versus adaptations is, in general, very domain- and case-specific and is beyond the scope of this paper.

VI. Related Work

Anomaly detection in time-series is a heavily studied topic within data science [21], [24]. In general, approaches can be categorized in supervised, which rely on labeled datasets and employ classification techniques such as support vector machines and decisions trees, and unsupervised, which employ clustering and statistical techniques [25]. Most approaches are partially online in that they use an offline phase to train a detection model whereas the actual detection can be done online, i.e. as the data come [25]. Our framework assumes no availability of labeled datasets and employs statistical methods with minimal assumptions on the underlying distributions, i.e. quartile analysis and modified z-score method. It also follows the idea of learning models offline and employing them online. In our case, offline learning includes the identification of the most effective seasonal pattern (we used a year's data in our experiments) and the determination of thresholds for the two different scores based on user input.

Anomaly detection has also a number of applications in software systems, the most prominent examples being the detection of performance regressions [26], [27] and detection of attacks and intrusions in network data [28]. Viewing an intrusion detection system as a self-adaptive system, anomaly detection comprises the analysis phase of its MAPE-K loop. We focus instead on using anomaly detection as a tunable mechanism to detect and filter wrong data or outliers that may bias the decisions of a self-adaptive system (such as a traffic control system) or cause inaccurate measurements or inflation of metric variance in online experimentation pipelines [4].

Looking at the literature of outlier detection methods, there are a number of tests that can be used when the number of outliers to be detected can be estimated by the user. For instance, Generalized ESD (Extreme Studentized Deviate) test [29], [30] requires an upper bound of number of outliers to be provided. Twitter has built a novel approach based on ESD to employ in its production long-term timeseries data [20], [26]. Another standard practice for outlier detection

using inter-quartile range employs 3X factor to mark potential outliers where the 3X factor is arbitrarily chosen [31], [32]. In practice, knowing or estimating the number of outliers in advance may be difficult. Also, the above methods cannot be tuned to incorporate domain knowledge in the anomaly detection process. In contrast, our framework (i) does not assume that a number of anomalies is provided upfront, (ii) relies on the first and third quartiles to assign scores to potential anomalies, and (iii) incorporates domain knowledge in the threshold selection process, the result of which determines the final anomalies.

Our proposal bears similarities to EGADS (Extensible Generic Anomaly Detection System), Yahoo's generic anomaly detection framework, internally used for detecting outliers in both system data and business key performance indicators [33]. Indeed, we also generate models of normal behavior against which we can perform detection in real time by calculating the deviation from the normal behavior and applying thresholds to judge when an anomalous behavior is to be reported. Contrary to them, we do not allow to plug in different methods for modeling timeseries (ARIMA, Exponential Smoothing, etc.), but employ a more lightweight, statistical-based approach which relies on seasonal patterns of the original timeseries and their differenced counterpart.

VII. Conclusion

In this paper, we described a novel framework for anomaly detection that is generic and tunable by user input. The main benefit of our approach is that our framework does not require an expert in data analytics and can thus be easily used by software architects, which have general domain knowledge. Within a self-adaptive system or a continuous experimentation pipeline, the framework can be used to increase trustworthiness in data-driven decisions. In particular, our framework can be used for learning the normal patterns of a timeseries-outputs of running systems and detect different types of anomalies that should be dealt with by the system to increase its trust in data-driven decisions.

Acknowledgment

This work has been sponsored by the German Ministry of Education and Research (grant no 01Is16043A) and by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the Centre Digitisation.Bavaria, as part of the Virtual Mobility World (ViM) project.

References

[1] E. Woods, "Software Architecture in a Changing World," *IEEE Software*, vol. 33, no. 6, pp. 94–97, Nov. 2016.

[2] D. Tang, A. Agarwal, D. O'Brien, and M. Meyer, "Overlapping experiment infrastructure: More, better, faster experimentation," in *Proc. of SigKDD 2010*, ACM, 2010, pp. 17–26.

[3] R. Kohavi *et al.*, "Online experimentation at Microsoft," in *Data Mining Case Studies and Practice Prize III*, 2009, vol. 11.

[4] Z. Zhao, M. Chen, D. Matheson, and M. Stone, "Online Experimentation Diagnosis and Troubleshooting Beyond AA Validation," in *Proc. of DSAA'16*, 2016, pp. 498–507.

[5] E. Bakshy, D. Eckles, and M. S. Bernstein, "Designing and Deploying Online Field Experiments," in *Proc. of WWW'14*, 2014, pp. 283–292.

[6] Y. Xu, N. Chen, A. Fernandez, O. Sinno, and A. Bhasin, "From Infrastructure to Culture: A/B Testing Challenges in Large Scale Social Networks," in *Proc. of KDD'15*, 2015, pp. 2227–2236.

[7] D. N. Hill, H. Nassif, Y. Liu, A. Iyer, and S. V. N. Vishwanathan, "An Efficient Bandit Algorithm for Realtime Multivariate Optimization," in *Proc. of the KDD'17*, 2017, pp. 1813–1821.

[8] "Uber Experimentation Platform," 18-Jun-2018. [Online]. Available: https://eng.uber.com/tag/experimentation/.

[9] D. I. Mattos, J. Bosch, and H. H. Olsson, "Challenges and Strategies for Undertaking Continuous Experimentation to Embedded Systems: Industry and Research Perspectives," in *XP 2018*, 2018, pp. 277–292.

[10] H. Holmström Olsson and J. Bosch, "Towards Data-Driven Product Development: A Multiple Case Study on Post-deployment Data Usage in Software-Intensive Embedded Systems," in *Lean Enterprise Software and Systems*, 2013, pp. 152–164.

[11] H. H. Olsson, J. Bosch, and A. Fabijan, "Experimentation that Matters: A Multi-case Study on the Challenges with A/B Testing," in *Software Business*, vol. 304, Springer, 2017, pp. 179–185.

[12] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A Contextual-bandit Approach to Personalized News Article Recommendation," in *Proc. of WWW'10*, New York, NY, USA, 2010, pp. 661–670.

[13] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google Vizier: A Service for Black-Box Optimization," in *Proc. of KDD '17*, 2017, pp. 1487–1495.

[14] M. Salehie and L. Tahvildari, "Self-Adaptive Software: Landscape and Research Challenges," *ACM TAAS*, vol. 4, no. 2, May, pp. 1–40, 2009.

[15] J. Kephart and D. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[16] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, 4th ed. Chapman & Hall/CRC, 2007.

[17] J. Gupchup *et al.*, "Trustworthy Experimentation Under Telemetry Loss," in *Proc. of CIKM'18*, Torino, Italy, 2018, pp. 387–396.

[18] "tsoutliers: Detection of Outliers in Time Series." [Online]. Available: https://cran.r-project.org/web/packages/tsoutliers/index.html.

[19] Twitter, Inc and other contributors, "AnomalyDetection R package." [Online]. Available: https://github.com/twitter/AnomalyDetection.

[20] J. Hochenbaum, O. S. Vallis, and A. Kejariwal, "Automatic Anomaly Detection in the Cloud Via Statistical Learning," *arXiv:1704.07706 [cs]*, Apr. 2017.

[21] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–58, Jul. 2009.

[22] B. Iglewicz and D. C. Hoaglin, *How to Detect and Handle Outliers*. ASQC Quality Press, 1993.

[23] D. C. Montgomery, *Introduction to Statistical Quality Control*, 6 edition. Hoboken, N.J: Wiley, 2008.

[24] D. Cheboli, "Anomaly Detection of Time Series," University of Minessota, 2010.

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

[26] O. Vallis, J. Hochenbaum, and A. Kejariwal, "A Novel Technique for Long-Term Anomaly Detection in the Cloud," in *6th USENIX Workshop on Hot Topics in Cloud Computing*, 2014.

[27] J. Ehlers, A. van Hoorn, J. Waller, and W. Hasselbring, "Self-adaptive software system monitoring for performance anomaly localization," in *Proc. of ICAC'11*, 2011, pp. 197–200.

[28] G. F. Cretu-Ciocarlie, A. Stavrou, M. E. Locasto, and S. J. Stolfo, "Adaptive Anomaly Detection via Self-calibration and Dynamic Updating," in *Recent Advances in Intrusion Detection*, vol. 5758, Springer, 2009, pp. 41–60.

[29] B. Rosner, "On the Detection of Many Outliers," *Technometrics*, vol. 17, no. 2, pp. 221–227, 1975.

[30] B. Rosner, "Percentage Points for a Generalized ESD Many-Outlier Procedure," *Technometrics*, vol. 25, no. 2, pp. 165–172, 1983.

[31] John W. Tukey, *Exploratory Data Analysis*. Addison-Wesley, 1977.

[32] J. Laurikkala, M. Juhola, and E. Kentala, "Informal identification of outliers in medical data," in *Proc. of Fifth International Workshop on Intelligent Data Analysis in Medicine and Pharmacology*, 2000.

[33] N. Laptev, S. Amizadeh, and I. Flint, "Generic and Scalable Framework for Automated Time-series Anomaly Detection," in *Proc. of the KDD'17*, 2015, pp. 1939–1947.