POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Master's Degree Thesis

Unsupervised Anomaly Detection on Multivariate Time Series in an Oracle Database

Supervisors
prof. Daniele Apiletti
ing. Davide Burdese
ing. Ilaria Varacalli

Candidate
Davide Di Mauro

Academic Year 2022-2023

Summary

This thesis presents an unsupervised anomaly detection method for multivariate time series in an Oracle database, a relational database management system widely used in various domains. Anomaly detection is a machine learning task that aims to identify patterns in data that deviate from the expected behavior, and has applications in database performance monitoring, fraud detection, and intrusion detection. However, most existing methods require labeled data or manual selection of training data, which are not feasible in a truly unsupervised scenario. The proposed method is based on adversarially trained autoencoders, which are neural networks that learn to compress and reconstruct the input data, and can detect anomalies by measuring the reconstruction error. The proposed application consists of two phases: a training phase, where the autoencoders are trained on normal data selected automatically from the database metrics; and an inference phase, where the model is exposed to new data and anomalies are identified using dynamic thresholds. The method is tested on data collected from different Oracle database instances and considering a variable number of database statistics. The results show that the proposed method can effectively detect multivariate anomalies and can be applied in a production environment thanks to its ability to detect anomalies in near real-time. The main contributions of this thesis are: (1) a novel unsupervised anomaly detection framework for multivariate time series in an Oracle database; (2) a general and scalable architecture for implementing the method; (3) a comprehensive evaluation of the method on real-world data.

Acknowledgements

First and foremost, I want to thank my parents and my grandparents in particular for their love and support. I was allowed to study carefree and understand the worth of knowledge because of the numerous sacrifices they made.

To my dear friends, thank you for always being there, listening to all my complaints. Your presence was a constant reminder to enjoy the journey and have a beer.

A dutiful thank you to all my colleagues, for welcoming me so warmly and for the constant help over this month, your role was fundamental for the outcome of this thesis.

Last but not least, I want to thank my wonderful partner Ester. I've always found strength and inspiration in your encouragement, tolerance, and faith in me. I have been supported by your love and understanding during the highs and lows of this academic journey and beyond. I am very lucky to have you by my side, and I am so grateful for your constant support during all this years.

In conclusion, I would like to thank everyone who has directly or indirectly helped with my thesis. Your support, and encouragement been essential, and I am grateful for that. Reaching this important milestone would not have been possible without you.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

We live in a world where almost everything depends on technology to work and this results in huge amounts of data being produced every day. Managing the data generated every day is an incredible challenge and it is evident the need for a stable and well-managed infrastructure. Of course, this is not an easy task, for both the complexity and criticality of these systems.

It is precisely within this context that the machine learning research field of anomaly detection can be appreciated. Anomaly detection consists in identifying patterns in data that significantly differ from the expected behavior, and this has a wide range of practical applications, from identifying early signs of disease in medical data to fault detection in manufacturing.

Back to the problem of managing a complex IT infrastructure, Anomaly Detection can become a useful tool for helping database administrators do their work in a small context, and it becomes a necessity in larger companies.

1.1 Company

The research work presented in this thesis is done at **Mediamente Consulting s.r.l.**, a consulting company operating in the IT sector, specialized on the topic of advanced business analytics. It was founded in 2013 as a innovative startup at the i3P incubator of Politecnico di Torino from which it exited in 2016, and now it was acquired by Var Group (SeSa S.p.A. group) as part of its Data Science business unit.

The company is composed of five business unit: Corporate Performance Management, Advanced Analytics, Business Intelligence, Data Integration and Management and Technological Infrastructure.
One of the main business cores of the company is the aforementioned Technological Infrastructure unit, which deals with a range of activities, from consulting activities on various Oracle technologies including engineered systems, to performance analysis and assessment of the architecture, to the management and updating of the technological infrastructure both “on premise”and in cloud. This thesis work was done in conjunction with the Technological Infrastructure business unit.

At the moment of writing the company is strongly specialized in the installation and maintenance of Oracle products, from the well known Oracle Database to the Weblogic Application Server or the Oracle Application Express (APEX).

1.2 Case Study

As explained in the previous section, after the installation of Oracle products, the main activity done by the infrastructure business unit, is monitoring the clients databases. This is done using an enterprise monitoring software, the AMS (Application Management System), based on the configuration of manually selected thresholds. When the database parameters monitored (e.g. CPU usage, aviable memory, etc...) cross the threshold an alert is send to the AMS and the operators, so that they can rapidly respond to the problem.

This approach, while simple, has many downsides:

- **Not all the alert received are synonyms of anomalies**. For example, an host CPU that will cross the threshold for a couple seconds for then return to normal values will be erroneously signaled from the system as an anomaly.
- **Identifying the threshold is a difficult task**, that requires years of experience and a profound knowledge of the client system and usage.
- When **the desired metric to monitor are hundreds for each database** instance of every client, it is almost impossible for a company like Mediamente Consulting to set thresholds for each metric (notice that every instance needs different threshold for the same metric), with the almost complete certainty of being submerged by useless alerts, drastically slowing down the operators' work.
- Considering each metric separately has the disadvantage of preventing being able to observe the behavior of several metrics simultaneously in order to identify an anomaly.

It is precisely in this context that is evident how the use of machine learning techniques can improve the company's monitoring system. This is precisely the reason that justifies the thesis work presented in the following pages.

1.3 Project Structure and Workflow

The timeline in Figure 1.1 represent the workflow adopted for this thesis.

The first phase was the study of the software and tool used by the company in order to understand not only how the Oracle's ecosystems works but also how the company manages their jobs. This last part was especially important to develop a solution that fitted well the company needs.

In parallel, a study of the problem of anomaly detection and the state-of-the-art techniques to address it was conducted.

The next phase was the data retrieval and exploration one, where the data of interest was collected from the database. One the data was retrieved, a thorough analysis of the data at hand, in conjunction with the company experts, was conducted. Their help is fundamental, since their experience is the most important judgment parameter to select the metric of interest, even more than statistical one.

I then moved to a cyclical phase in which I implemented and tested the different algorithms, to choose which implement.

Finally I developed and documented an application, paying attention in making it versatile (new algorithms can be implemented by other employees and also is compatible with different data from the one used in this work) and future-proof since new modules can easily be added increasing the app functionalities.

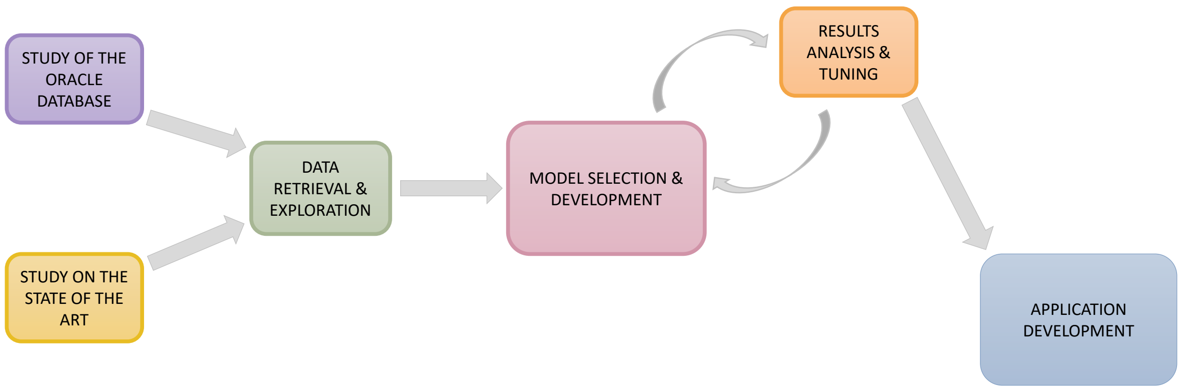

STUDY OF THE ORACLE DATABASE
STUDY ON THE STATE OF THE ART
DATA RETRIEVAL & EXPLORATION
MODEL SELECTION & DEVELOPMENT
RESULTS ANALYSIS & TUNING
APPLICATION DEVELOPMENT

Figure 1.1: Adopted workflow

Project Structure

The subsequent chapters are divided as follows:

- In chapter 2 an introduction of the anomaly detection problem, a description of the Oracle Database and the view considered is given.
- chapter 3 contains a brief description of the theoretical background needed to understand the proposed solution.

- chapter 4 describes in detail the main algorithms used in this project.
- In chapter 5 is described and argued the proposed solution.
- The results obtained during different test are described and compared in chapter 6.
- Finally, chapter 7 deals with the conclusions and indicates the possible future developments of the project.

Chapter 2

Background

2.1 Anomaly Detection

Anomaly detection is the task of finding patterns in data with anomalous behavior compared to the majority of instances. The first thing to do is define what are anomalies; Hawkins defines an outlier as: “an observation which deviates so significantly from other observations as to arouse suspicion that it was generated by a different mechanism”Hawkins [1980] while Maddala defines it as: “a data point that differs significantly from other observations”Maddala [2001].

There are different fields of application for anomaly detection:

- Medical diagnosis
- Fraud detection
- Intrusion detection
- Manufacturing defects detection
- Network analysis

For all this applications, different techniques can be found in literature, starting from more statistical approaches, moving on to linear models and clustering, and ending up with newer techniques like neural networks and autoencoders.

2.1.1 Type of Anomalies

A more precise distinction can be drown, in fact we can identify 3 different types of outliers:

- **Point outlier**: when a single point has a value that is anomalous compared the entirety of the data.
- **Contextual outlier**: is an instance that has abnormal values with respect to its context and not globally. Finding this type of outliers represent a more challenging task compared to point anomalies.

- **Collective outlier**: a set of data point is considered a collective outlier if the set significantly deviates from the norm, but the points of the set are not anomalous in either the context of the set or the global one. For example the individual observations may not be anomalies, but their co-occurrence make them so.

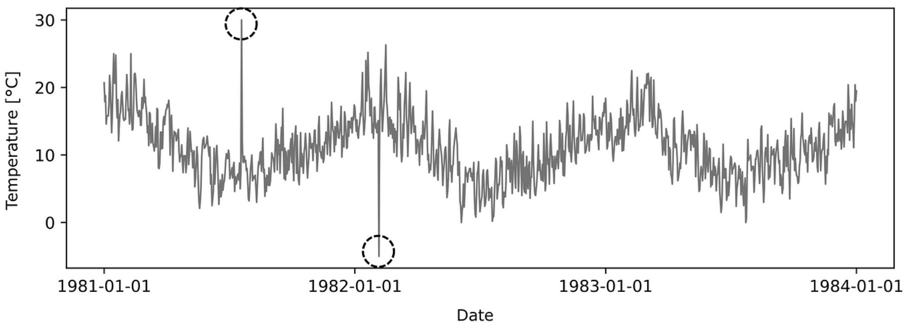


Figure 2.1: Point anomaly

2.1.2 The Unsupervised Setting

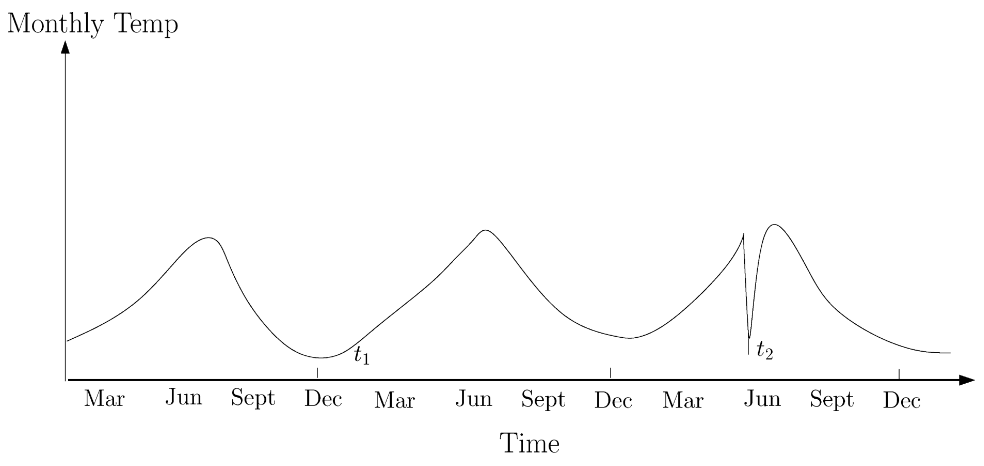
One of the first things to consider when engineering a solution to a machine learning problem is to understand the available data and the presence of labels, in order to choose a supervised or an unsupervised approach. Anomaly detection is usually an unsupervised task since it deals with a huge quantity of data, that grows over time, and the anomalies usually are rare events, thus making it unreasonable to label the data, especially in a context where the only way to identify anomalies is thanks to domain expert and this makes the labeling of data an expensive and time-consuming activity.

Excluding a priori the possibility of having labeled data, I opted for an unsupervised approach. The idea behind unsupervised learning is to discover hidden patterns in data, hence creating a model able to detect if a new given set of values contains anomalies or not. This comes with various challenges, such as:

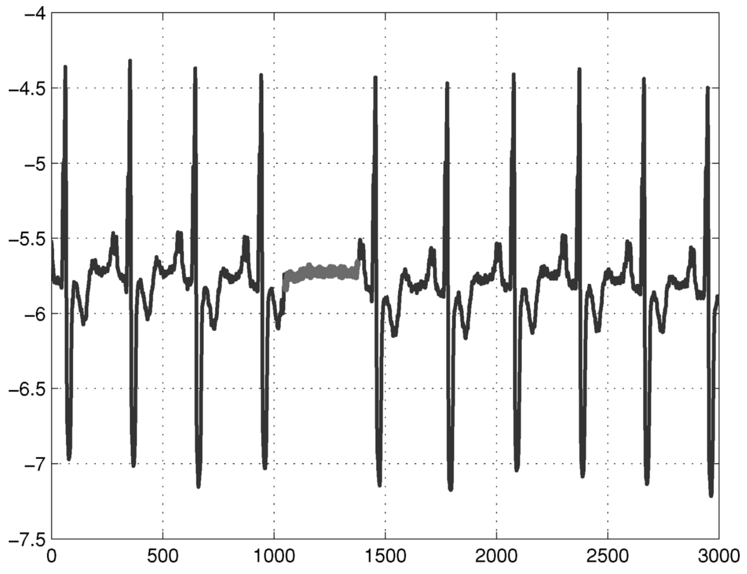
- Higher inaccuracy
- Difficulties in validating the results, usually needing the intervention of domain experts
- Computational complexity due to the huge amount of data to analyze
- It is difficult to understand the basis for which some values are considered anomalies or not

2.2 Oracle Database

Oracle Database is a relational database management system (RDBMS) produced by Oracle Corporation.



(a) Contextual anomaly



(b) Collective anomaly

Figure 2.2: Different type of anomalies Chandola et al. [2009]

An Oracle server is composed by at least two entities:

- **Database**, that is is an organized collection of structured data, more specifically the psychical file where the data is stored.
- **Instance**, that is to say the combination of the memory areas and background processes that make up the running installation of the system (a database could exist even without an instance, but it would be unusable since there would be no way to access to the data stored in it).

In order to access to the file in a database, the instance must be associated to the physical database. An instance can access only one database at a time, but multiple instances can access simultaneously to the same database.

A representation of the Oracle Database architecture is provided in Figure 2.3

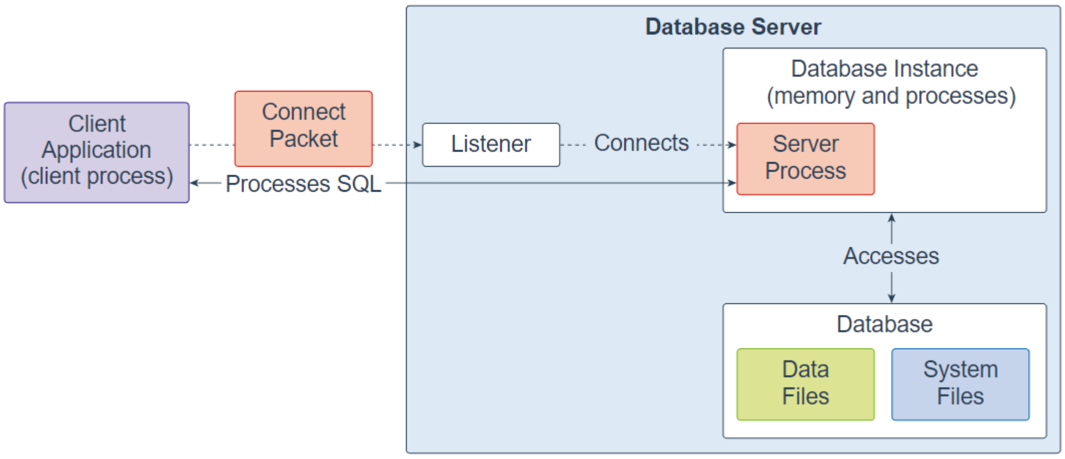


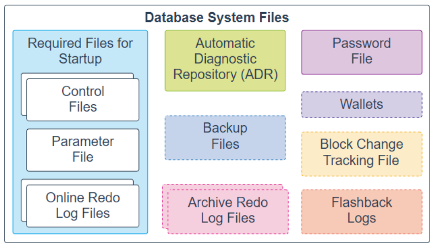
Figure 2.3: Oracle Database Architecture oracletutorial.com [2023]

Database

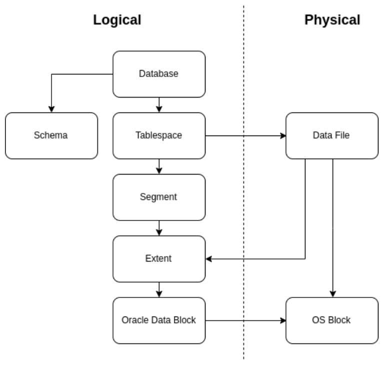
The data in the Oracle Database is stored following a physical and logical schema:
The physical schema (Figure 2.4a) represent how the data is actually stored in the database: the most important files stored are the *data files*, containing the real data of the client and the data of the logical structures of the database (i.e. tables, indexes, etc...). Other than the data files, there are also *control files* containing metadata describing the physical structure of the database and *online redo log files* recording all the changes done to the database.
The logical storage structure are instead used for fine-grained control of disk space usage. The simpler logical unit is the *data block*, corresponding to a number of bytes on the disk; logically contiguous data blocks form what is called an *extent*. A set of extents allocated to store objects like tables for example, are called *segments*. Multiple segments are stored into logical containers called *tablespaces*.

Instance

An instance is the interface that lets client applications connect to the database. It consist of three main parts: the *system global area* (SGA), *program global area* (PGA) and background processes (Figure 2.5). When the instance is started, the SGA memory is allocated and shared between all processes, while a PGA is allocated to each process when is created and released when it ends.



(a) Oracle Database physical storage structure oracletutorial.com [2023]



(b) Oracle Database logical storage structure oracle world.com [2023]

Figure 2.4: Oracle Database storage structures

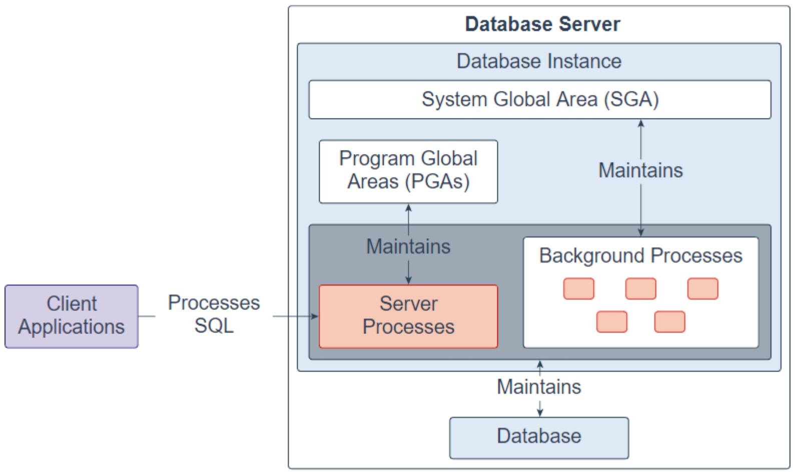


Figure 2.5: Oracle Instance oracletutorial.com [2023]

2.3 V$SYSMETRIC

The Oracle Database automatically collects all the information about the performance metrics of the database, sampling them at 15 or 60 seconds intervals. The workload metrics are historicized in the system's views *V$SYSMETRIC*, *V$SYSMETRIC_HISTORY* and *V$SYSMETRIC_SUMMARY*. The view *V$SYSMETRIC_HISTORY* contains the data of the last hour, while the *V$SYSMETRIC* contains only the data of the last sample. In order to collect the data for a long period of time, the strategy adopted was to write a simple script to store the data of the *V$SYSMETRIC* each minute into a new table.

It is important to mention that Oracle offers other solution to collect performance data, the most common one is the Automatic Workload Repository (AWR), a tool that collects, processes, and maintains performance statistics for the database offering performance reports and views to consult. The choice made to use the *V$SYSMETRIC* view is due to the fact that this table is available in all installation of the Oracle Database, while the AWR requires an Enterprise subscription and the further purchase of the Diagnostic Pack and not every client can afford it. With the idea of creating a tool that the company could use for every client this was a forced but necessary choice.

This view contains values about 155 different metrics, that can be logically divided into 8 groups:

Group 1	Group 2	Group 3	Group 4	Group 5	Group 6	Group 7	Group 8
CPU	**I/O**	**CACHE**	**SQL**	**DEBUG**	**RAC**	**USERS**	**ENQUEUE**

Table 2.1: *V$SYSMETRIC* metric groups

During the research conducted, an extensive amount of time was dedicated to the selection of the metrics to use. It was decided in conjunction with the company that an empirical approach, based on testing and, above all, on the experience of the company engineers was preferable instead of a purely statistical one.

A final set of 19 metrics was selected (Table 2.2), since these are the one that multiple sector expert considered critical to monitor.

While not all experts suggested the same metrics, it was clear from the start that the most important one was the **Database Time Per Sec** or ***DB time***. This metric is the amount of elapsed time in microseconds that the database spends performing user-level calls, excluding the time spent on background processes. What it means is that the *DB time* corresponds to the total time spent by user processes either actively working or actively waiting in a database call. It is defined as:

$$DB\ time = CPU\ time + I/O\ time + non\text{-}idle\ \ Wait\ time \tag{2.1}$$

Since the *DB time* represents the general level of stress of the database, it is the first metric that a database administrator looks when searching for problems, this metric was essential in validating the results of the proposed solution. Of course analyzing only the *DB time* is not enough, since it is only the “symptom”and not the actual cause of the problem.

Group 1 **CPU**	Group 2 **I/O**	Group 3 **CACHE**
CPU Usage Per Sec	*DB Block Changes Per Sec*	*Consistent Read Gets Per Sec*
Host CPU Usage Per Sec	*I/O Megabytes per Second*	*DBWR Checkpoints Per Sec*
	Physical Reads Per Sec	*Logical Reads Per Sec*
	Physical Writes Per Sec	*Redo Generated Per Sec*

Group 4 **SQL**	Group 5 **DEBUG**	Group 6 **RAC**
Average Active Sessions	*Database Time Per Sec*	
Executions Per Sec		
Hard Parse Count Per Sec		

Group 7 **USERS**	Group 8 **ENQUEUE**
Logons Per Sec	*Enqueue Waits Per Sec*
User Calls Per Sec	
User Commits Per Sec	
User Rollbacks Per Sec	

Table 2.2: Selected metrics from the *V$SYSMETRIC* view

Chapter 3

Related Works

3.1 Time Series

The work presented in this thesis revolves around the analysis of time series data, since the database's metrics change over time, hence the need of the following definitions.

A time series can be defined as a set of data points of the same entity collected at regular intervals; we can identify two different types of time series:

3.1.1 Univariate Time Series

Using the definition provided by Blázquez-García et al. [2020], a *univariate time series* $X = \{x_t\}_{t \in T}$ is an ordered set of real value observation, where each observation is recorded at a specific time $t \in T \subseteq \mathbb{Z}^+$ and x_t is a realization of a certain random variable X_t

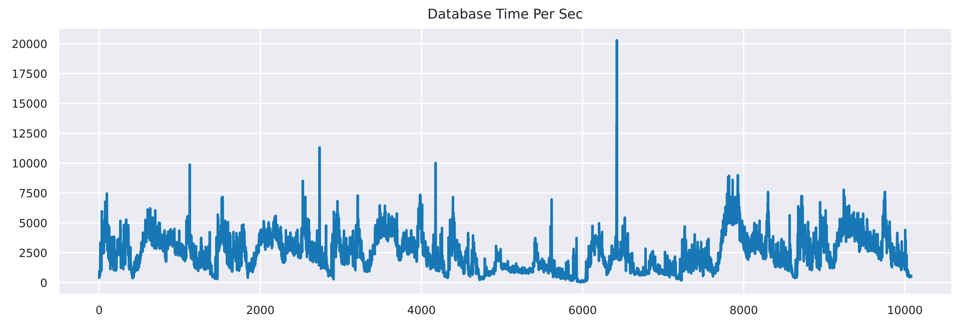


Figure 3.1: Example of univariate time series analyzed in this thesis.

3.1.2 Multivariate Time Series

Using the same notation, a *multivariate time series* $X = \{x_t\}_{t \in T}$ is defined as an ordered set of k-dimensional vectors, each sampled at a time $t \in T \subseteq \mathbb{Z}^+$, consisting of k

observations $x_t = (x_{t_1}, x_{t_2}, ..., x_{t_k})$.

For each dimension $j \in \{1, 2, ..., k\}$, x_t is a realization of a time-dependent random variable X_{jt} in $_t = (X_{t_1}, X_{t_2}, ..., X_{t_k})$. The complexity in analyzing a multivariate case resides not only in the multi-dimensionality of the data, but mostly on the fact that each variable could not only depend on its past values but also on the other time-dependent variables.

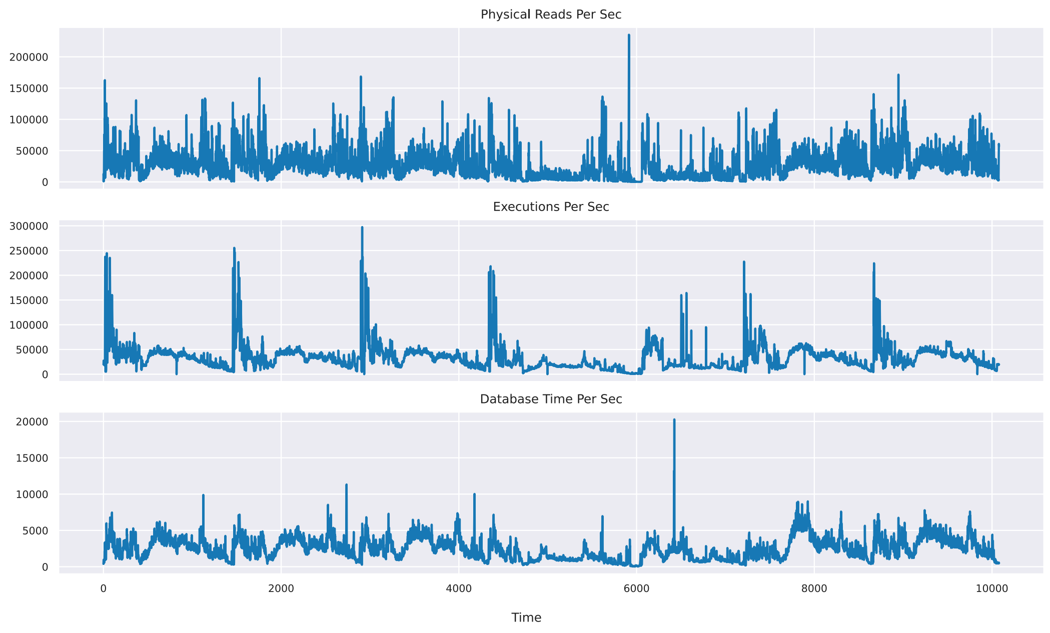


Figure 3.2: Multiple database's metrics make up a multivariate time series

3.1.3 Time Series Decomposition

Time series decomposition means to decompose into different components a time series. The most common approach is the decomposition based rates of change, which consists in dividing the time series into three components:

- T_v: The *trend*, that represent the long-term behavior of the data, that is if the mean μ is not constant but changes over time.
- S_v: The *seasonality*, that reflect the seasonal patterns of the time series. These pattern emerge when the data is influenced by seasonal factors like the day of the week or time of the year.
- R_v: The *reminder*, which is the most relevant component in anomaly detection, it consists in residuals of the time series, after the other components are computed. This part collects everything that is left over from the "regular" pattern, like the outliers that we want to identify, possibly improving the performance of the implemented models.

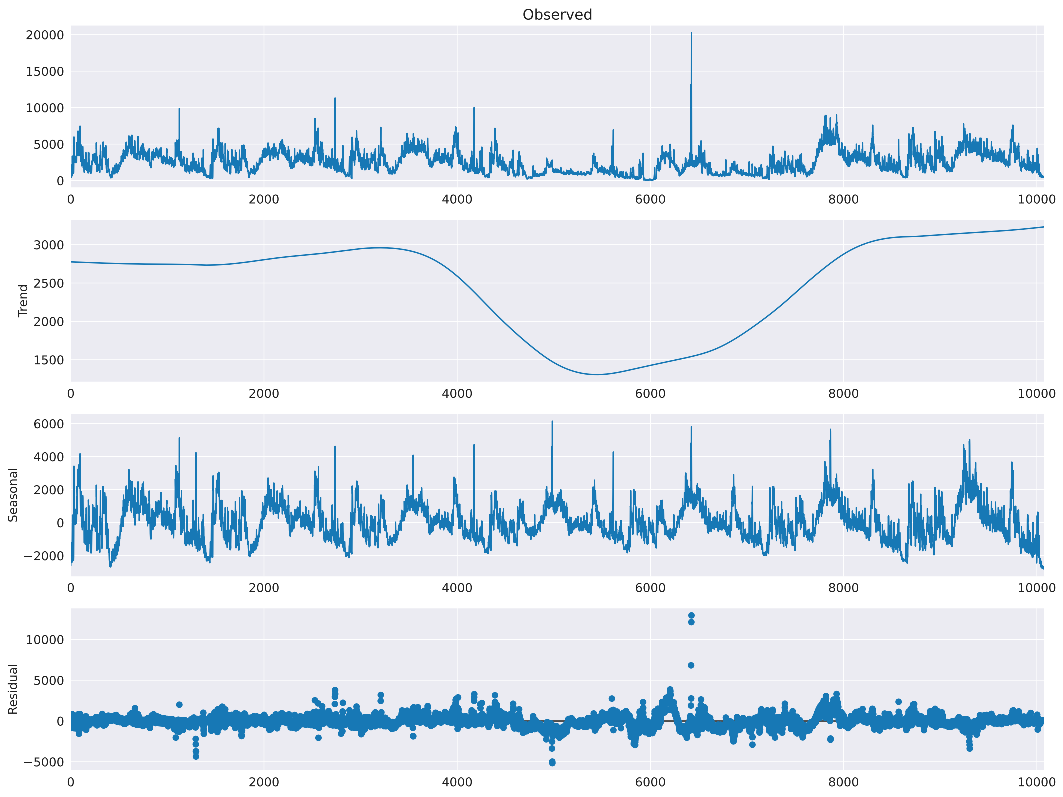


Figure 3.3: Multiple database's metrics make up a multivariate time series

3.2 Autoencoders

Autoencoders are a feed-forward neural network, trained to reproduce its input at the output layer. They learn in an unsupervised manner the most relevant aspect of the observable data extracting its most important features and creating a compressed representation of the input data. The structure of this network is made by two components:

- The encoder E, which learns an encoding function and transforms the input data, reducing its dimension
- The decoder D, that thanks to the decoded function tries to reconstruct the original data from the encoded one

As formally defined in Baldi [2011], the input X is transformed by the encoder $E : \mathbb{R}^n \to \mathbb{R}^p$ into a set of latent variables Z, and the objective of the decoder $D : \mathbb{R}^p \to \mathbb{R}^n$ is to transform Z back into the input space as the reconstruction R minimizing the reconstruction error (3.1), defined as the difference between the original input and the reconstructed output. Using the ℓ_2-norm as a regularization method to avoid overfitting to the identity function we have:

$$\mathcal{L}_{AE} = \|X - AE(X)\|_2 \tag{3.1}$$

where

$$AE(X) = D(Z)$$
$$Z = E(X)$$

Usually E and D are neural network, If both of them are linear operations, we have a linear autoencoder. If the non-linear operations are also removed the autoencoder obtains the same latent representation as in the Principal Component Analysis (PCA) Plaut [2018] thus making the autoencoders a generalization of this technique.

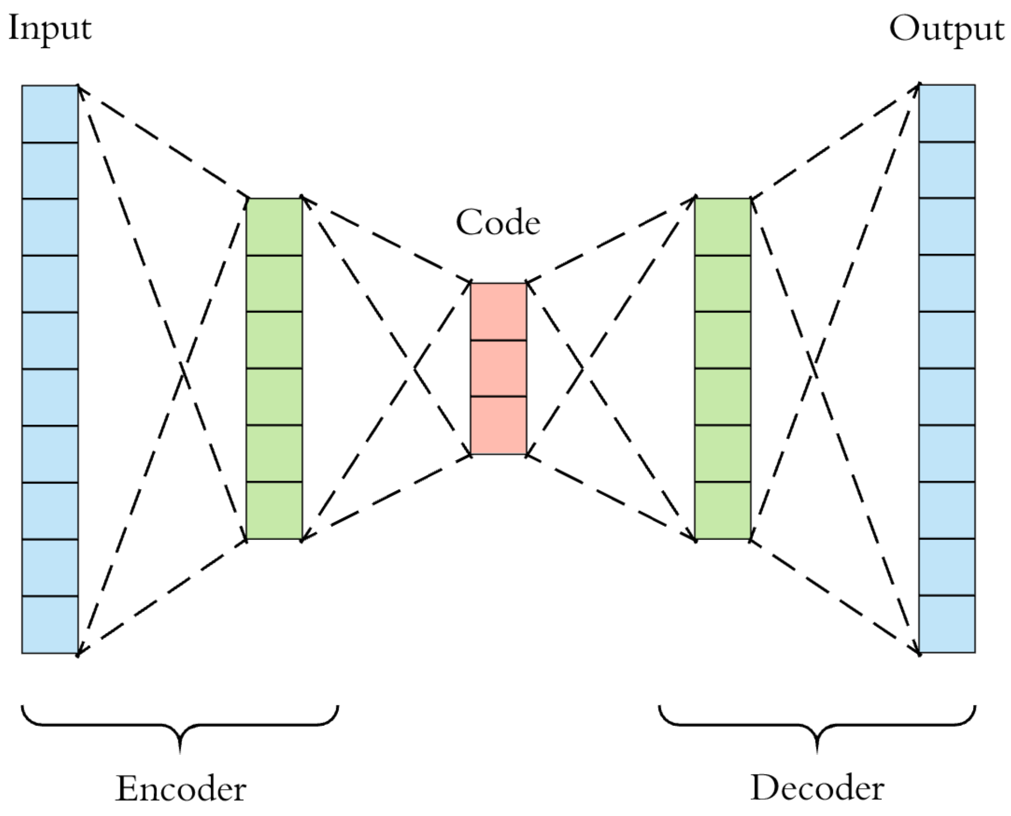


Figure 3.4: Example of autoencoder Dertat [2017]

There are several other types of autoencoders, as summarized in Bank et al. [2021]:

- Regularized autoencoders
- Sparse autoencoders
- Denoising autoencoders
- Contractive autoencoders
- Variational autoencoders

with several applications, like classification, clustering, **anomaly detection**, recommendation systems and more.

In the context of anomaly detection, an autoencoder can be used to learn a representation of the normal data, and when data containing anomalies is passed to the trained model, the reconstruction error will be greater then normal, hence identifying an anomaly. All this under the assumption that **only normal data is used for training**. This is a problem in a truly unsupervised scenario where is too expensive or even impossible to manually select the data for training; the inability to adhere to this assumption will be the basis for the model selection of the solution proposed in this thesis (USAD).

3.3 Generative Adversarial Networks

Generative adversarial networks (GANs) Goodfellow et al. [2014] are a machine learning technique used in both semi-supervised and unsupervised scenarios. They consists on a pair of networks, trained in an adversarial fashion, in competition between each other. An analogy often used to describe this kind of networks is the one where one network is an art forger and the other an art expert: the forger tries to trick the art expert creating realistic images, while the art expert tries to distinguish the counterfeits from the originals.

They are constituted by two separate sub-modules, a generator G and a discriminator D. it is important to emphasize that the generator has no access to the input data, and the only may for it to learn is trough the interaction with the discriminator.

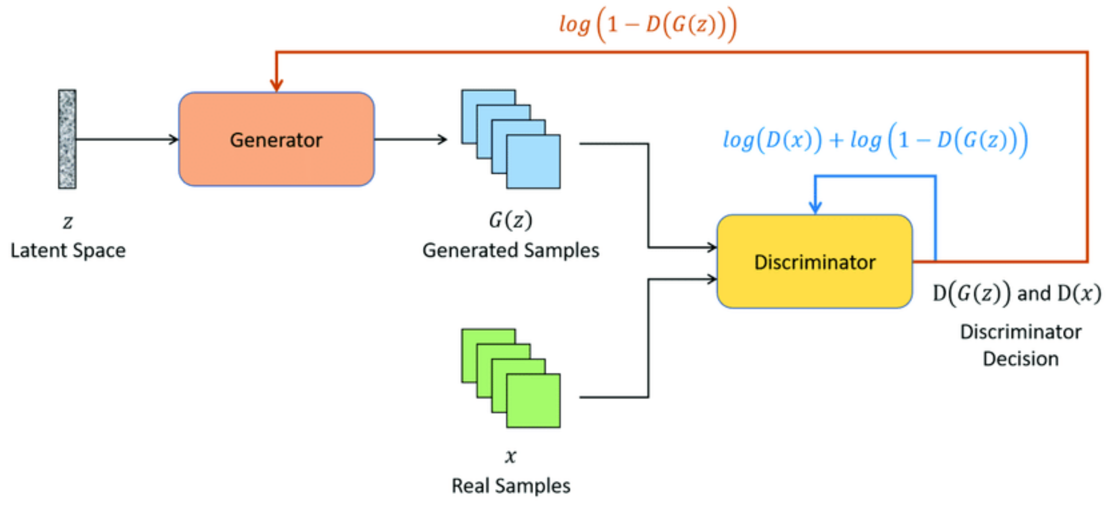


Figure 3.5: Typical generative adversarial network (GAN) architecture Vint et al. [2021].

Formally, the decoder D is trained to maximize the probability of assigning correctly the label for both training samples and the ones generated by G. Simultaneously G is trained to minimize $\log(1 - D(G(z)))$ where in general $G(x)$ represent the probability of x coming from the real data rather then from the fake one.

This makes it possible to define this task as a minmax optimization problem with a value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(x)}[\log(1 - D(G(z)))] \tag{3.2}$$

where p_{data} is the probability distribution of the original data, and p_z is the one of the data produced by the generator. It is easy to see that the generator G is optimal when the discriminator D is maximally confused, hence when $p_{data}(x) = p_z(x)$, which is the same as having the discriminator predicting 0.5 for all samples from x. Notice that this equilibrium is very difficult to reach, since the increase of the generator cost function can cause the decrease of the discriminator cost function and vice versa, making the GANs unstable.

Another issue with GANs is the mode collapse problem; it consists of the inability of the network to diversify, since the generator only objective is to trick the discriminator, not representing the the multi-modality of the real data Che et al. [2017].

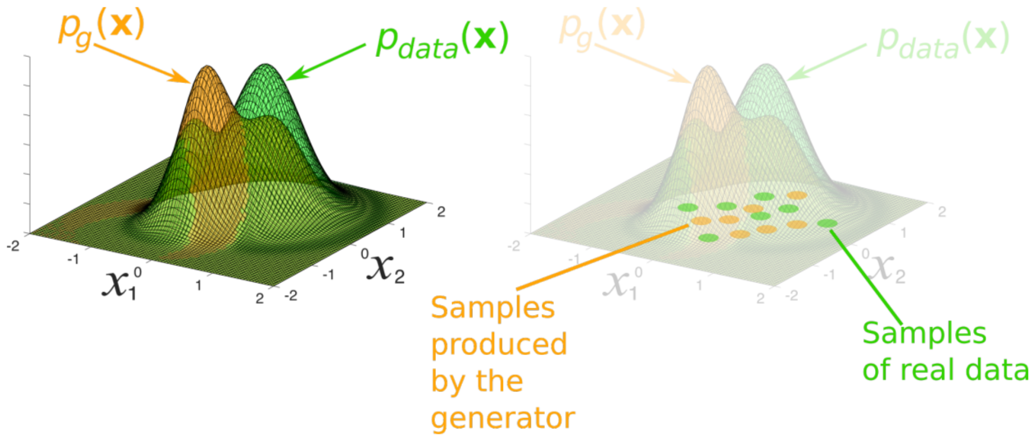


Figure 3.6: Probability distribution of the real data and the generated one, in the optimal case they will coincide Creswell et al. [2018].

GANs are a brilliant idea with a lot of applications, such as:

- Classification and Regression
- Image Synthesis
- Image-to-Image Translation
- Text-to-Image Translation
- Super-resolution
- Object Detection
- Image Implanting
- Image Blending

Chapter 4

Methods

This chapter is dedicated to the description of the two main techniques adopted in the solution proposed in this thesis. The practical implementation of these algorithms will be described in detail in the next chapter, in the sections below, a theoretical description of the algorithms is presented.

4.1 USAD

The UnSupervised Anomaly Detection for multivariate time series (USAD) Audibert et al. [2020] is a state-of-the-art method based on adversely trained autoencoders, used to detect anomalies in an unsupervised way, with a focus on the time and resources needed for training.

The idea is to use an encoder-decoder architecture in an adversarial training framework, combining the advantages of both techniques (described in the previous chapter), while compensating their limitation, especially the problem in training stability typical of generative adversarial networks.

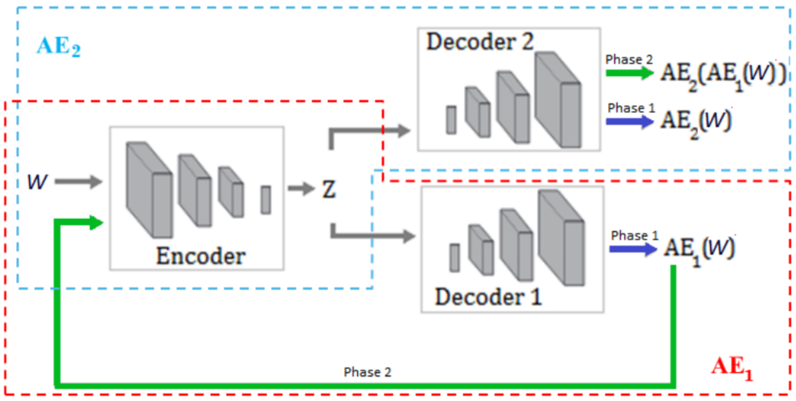
The method consists in an autoencoder architecture within a two phase adversarial framework: this is done with the objective of overcoming, thanks to the adversarial framework, the limitation of autoencoders (Autoencoders) consisting in the need of using normal data to train, while gaining stability from the usage of autoencoders instead of GANs, that suffers the problem of non-convergence and mode collapse (Generative Adversarial Networks).

4.1.1 Architecture

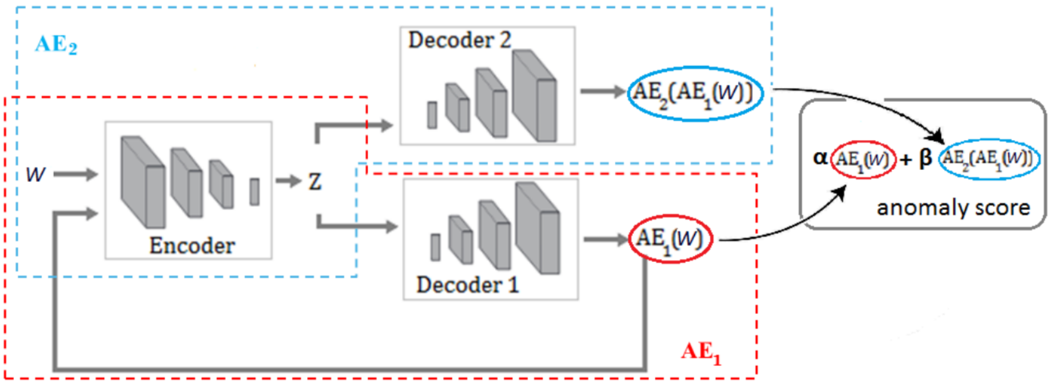
The USAD architecture is composed of three components: one encoder E and two decoders D_1 and D_2, forming two autoencoders, AE_1 and AE_2, with the same structure, i.e. a simple neural network.

$$\begin{aligned} AE_1(W) &= D_1(E(W)) \\ AE_2(W) &= D_2(E(W)) \end{aligned} \tag{4.1}$$

where W is a window of length K containing the values for each of the m variables of the multivariate timeseries ($m = 1$ is the univariate case).



(a) USAD training flow



(b) USAD detection flow

Figure 4.1: USAD architecture Audibert et al. [2020]

4.1.2 Two-phases training

The architecture is trained in two different steps:

1. **Autoencoder training:** in this phase the two autoencoders AE_1 and AE_2 learn to reconstruct the windows W from the "normal" data. Each encoder E compresses the information contained in W to the latent space Z, then the decoders D will reconstruct each compressed representation into the original space. This is described in the following equations representing the training losses (4.2):

$$\begin{aligned}\mathcal{L}_{AE_1} &= \|W - AE_1(W)\|_2 \\ \mathcal{L}_{AE_2} &= \|W - AE_2(W)\|_2\end{aligned} \tag{4.2}$$

2. **Adversarial training:** during this second phase, in an adversarial fashion, AE_1 will try to trick AE_2, while AE_2 objective is to learn to understand whether or not the data is coming from the original data W or is the fake one from AE_1.

   The data coming from AE_1 is compressed another time by E to the latent space Z, to then be reconstructed by D_2.

   The learning objective is therefore formulated as a minmax problem, where AE_1 tries to minimize the difference between W and AE_2's output, while the goal of AE_2 is to minimize such difference.

$$\min_{AE_1} \max_{AE_2} \|W - AE_2(AE_1(W))\|_2 \tag{4.3}$$

   where the following equations describe the training objective of each autoencoder:

$$\begin{aligned} \mathcal{L}_{AE_1} &= +\|W - AE_2(AE_1(W))\|_2 \\ \mathcal{L}_{AE_2} &= -\|W - AE_2(AE_1(W))\|_2 \end{aligned} \tag{4.4}$$

Summarizing, in the USAD architecture, the autoencoders have a double function:

- ***AE*₁**:

  phase 1 $\rightarrow$ **minimize** the reconstruction error

  phase 2 $\rightarrow$ **minimize** the difference between W and reconstructed output

- ***AE*₂**:

  phase 1 $\rightarrow$ **minimize** the reconstruction error

  phase 2 $\rightarrow$ **maximize** the reconstruction error of the reconstructed data by AE_1

Combining equations (4.2, 4.4) and expressing them also in function of the training epochs n we obtain:

$$\begin{aligned} \mathcal{L}_{AE_1} &= \frac{1}{n}\|W - AE_1(W)\|_2 + \left(1 - \frac{1}{n}\right)\|W - AE_2(AE_1(W))\|_2 \\ \mathcal{L}_{AE_2} &= \frac{1}{n}\|W - AE_2(W)\|_2 - \left(1 - \frac{1}{n}\right)\|W - AE_2(AE_1(W))\|_2 \end{aligned} \tag{4.5}$$

The important difference with generative adversarial networks is in the fact that ***AE*₂ doesn't act as a true discriminator**, because based on which input it receives the objective function that intervenes is different: the loss from equation 4.2 is used when the input is the original data, equation 4.4 instead intervenes when the input is the reconstruction.

The training algorithm is described in Algorithm 1

Algorithm 1 USAD training

```
**Input:** $\mathcal{W} = \{W_1, ..., W_T\}$, $N$ epochs
**Output:** Trained $AE_1$, $AE_2$
  $n \leftarrow 1$
  $E$, $D_1$, $D_2 \leftarrow$ init weights
  **while** $n < N$ **do**
      **for** $t = 1$ to $T$ **do**
          $Z_t \leftarrow E(W_t)$
          $W_t^{1'} \leftarrow D_1(Z_t)$
          $W_t^{2'} \leftarrow D_2(Z_t)$
          $W_t^{2''} \leftarrow D_2(E(W_t^{1'}))$
          $\mathcal{L}_{AE_1} \leftarrow \frac{1}{n}\|W_t - W_t^{1'}\|_2 + \left(1 - \frac{1}{n}\right)\|W_t - W_t^{2''}\|_2$
          $\mathcal{L}_{AE_2} \leftarrow \frac{1}{n}\|W_t - W_t^{2'}\|_2 - \left(1 - \frac{1}{n}\right)\|W_t - W_t^{2''}\|_2$
          $E$, $D_1$, $D_2 \leftarrow$ update weights
      **end for**
      $n \leftarrow n + 1$
  **end while**
```

4.1.3 Inference

At inference, the anomaly score of the model, that is a score describing the probability of a new window $\hat{W}$ to be an anomaly, is defined as:

$$\mathcal{A}(\hat{W}) = \alpha \|\hat{W} - AE_1(\hat{W})\|_2 + \beta \|\hat{W} - AE_2(AE_1(\hat{W}))\|_2 \tag{4.6}$$

The hyperparameters α and β must respect the condition $\alpha + \beta = 1$ and their function is to parameterize the trade-off between true positives and false positives. Increasing α will decrease the number of false positives (*low sensitivity scenario*), while increasing β will have the opposite effect, raising the number of false positives (*high sensitivity scenario*). Is important to notice that this parameters can be changed on-the-fly at runtime, giving the operators the possibility to tune the sensitivity of the algorithm based on their workload.

The inference algorithm is described in Algorithm 2:

Algorithm 2 USAD inference

```
Input: $\hat{\mathcal{W}} = \{\hat{W}_1, ..., \hat{W}_{T}*\}$, $\alpha$, $\beta$, threshold $\lambda$
Output: Labels $y = y_1, ..., y_T*$
  for $t = 1$ to $T*$ do
      $Z_t \leftarrow E(W_t)$
      $\hat{W}_t^{1'} \leftarrow D_1(E(\hat{W}))$
      $\hat{W}_t^{2''} \leftarrow D_2(E(\hat{W}_t^{1'}))$
      $\mathcal{A} \leftarrow \alpha \|\hat{W} - \hat{W}_t^{1'}\|_2 + \beta \|\hat{W} - \hat{W}_t^{2''}\|_2$
      $E$, $D_1$, $D_2$ $\leftarrow$ update weights
      if $\mathcal{A} \geq \lambda$ then
          $y_t \leftarrow 1$
      else
          $y_t \leftarrow 0$
      end if
  end for
```

4.2 KNN

The *k*-nearest neighbors algorithm (*k*-NN) is a simple method mainly used for classification and regression tasks.

For classification the algorithm works computing, for each data point to classify in a multidimensional feature space, the distance with the *k*-th neighbors, and then assigning to the new data point the label of the majority of the nearest points.

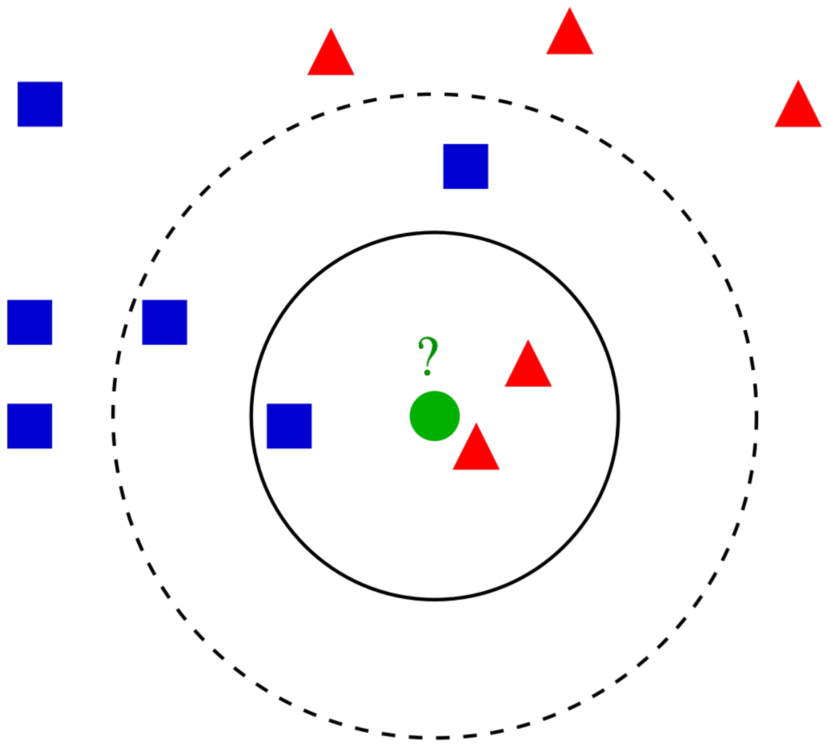


Figure 4.2: "Example of k-NN classification. The test sample (green dot) should be classified either to blue squares or to red triangles. If k = 3 (solid line circle) it is assigned to the red triangles because there are 2 triangles and only 1 square inside the inner circle. If k = 5 (dashed line circle) it is assigned to the blue squares (3 squares vs. 2 triangles inside the outer circle)"Wikipedia [2023].

The first step is to define how to compute the distance between two points in order to form decision boundaries. There are different ways to do it, the most common one are the following:

- **Euclidean distance**: arguably the most common one, the euclidean distance measures a straight line between two points.

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2} \tag{4.7}$$

- **Manhattan distance**: this distance is measured in terms of sum of absolute differences of the Cartesian coordinates between two points.

$$d(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i| \right) \tag{4.8}$$

- **Minkowski distance**: a generalization of the aforementioned distance, the Euclidean and the Manhattan one, where the parameter p allows for the definition of other distances. For example, with $p = 1$ it corresponds to the Manhattan distance, with $p = 2$ to the Euclidean distance and for $p \longrightarrow \inf$ it corresponds to the Chebyshev distance.

$$d(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i| \right)^{\frac{1}{p}} \tag{4.9}$$

Another important topic to discuss is the computational complexity of this method and the techniques to improve it.

The less efficient is the brute force one, where you have to compute all the distances to every data point, sort them and then take the k nearest. This has no computation during training $\mathcal{O}(1)$, but at inference the complexity is $\mathcal{O}(n \times d \times k)$ where n is the number of data points and d is the dimension of the data space.

To improve performance reducing the search space of distances to compute, some techniques based on trees were introduced, with the most popular being the k-d tree (Bentley [1975]) and ball tree (Omohundro [1989]) methods. This two techniques allows to split the search space using trees and computational geometry in order to allow for a faster search of the nearest points. In these cases, during training we have to compute the aforementioned trees, with a complexity of $\mathcal{O}(d \times n \log n)$ and a prediction time complexity of $\mathcal{O}(k \times \log n)$

4.2.1 KNN outlier detection

Even if it is not the most used application, the k-NN algorithm can be easily adapted in an unsupervised scenario, with the objective of detecting anomalies (Angiulli and Pizzuti [2002]). In fact, the distance to the k-th neighbor can be seen as a local density estimate, that is a common outlier score used in anomaly detection. For a data point, the larger the distance to the neighbors, the lower the local density, hence it is more likely that the point is an anomaly.
The problem is thus formulated:

"Given an input data set with N points, parameters n and k, a point p is a $D^k(p)$ outlier if there are no more than $n-1$ other points p' such that $D^k(p') > D^k(p)$"Ramaswamy et al. [2000].
In this definition, $D^k(p)$ denotes the distance of a point p from its k-th nearest neighbor.

Chapter 5

Proposed Solution

The objective of this thesis is to realize an application with a double function:

- a companion app for the operators used to analyze historical data, finding anomalies in the data and obtain images and other information to write reports
- a tool for near-real-time database performance monitoring, with the idea of integrating in the company workflow, to notify the operator when an anomaly is detected

In the following chapter a thorough description of the application is provided, and the result of three different experiments are presented (section 6.2, section 6.3 and section 6.4).

5.1 General Architecture

The architecture of the proposed solution is the one shown in Figure 5.1 and 5.2.

As described in section 4.1 the at the core of the application there is the USAD model. The model training pipeline is the most complex process of the application and is composed of the following steps:

1. **Training data selection**: The USAD architecture uses the adversarial training framework to compensate the intrinsic problem of autoencoders consisting in the need for only “normal”data during training in order to learn the normal data distribution and identifying anomalies when new data is passed. While is proven by the authors of the model the improved tolerance to noise of the model, in a complex scenario like the one analyzed, to obtain acceptable performance more consistently, a first screening of the data to remove the easily identifiable anomalies is needed. This can be done in two ways:

    (a) **Manual data selection**: this is straight forward; if the operator after an analysis is able to identify the critical time-spans to analyze can discard them and use the other data to train the model. Another possible solution is to use the customer reports to identify the anomalies, since it can be argued that an anomaly is effectively a problem only when the customer notices it.

(b) **Automatic data selection**: in order to facilitate the work of the operator but especially to create an application deployable in the client databases with the least amount of maintenance needed an automatic approach is required. As will be explained in section 5.3 this is done analyzing the *DB time* (section 2.3) and applying the k-NN method (section 4.2) to find anomalies. The Pythresh library implementation of the Yeo-Jonson transformation thresholder (subsection 5.4.1) to automatically selecting the threshold to discern an anomaly from a regular value.

2. **Pre-processing**: The pre-processing techniques used are the *z-score* normalization and the *min-max* scaling. The first is used to ensure that data has zero mean and unit standard variation, while the second is used to scale the values in the interval [0,1].

3. **Windows creation**: In order to keep information about the "temporal evolution"of the data, USAD groups the value points into windows of size K, to then predict if the window is an anomaly or not. The windows are created collecting K vector of size m metrics; then the widows slides by one position, creating a window with only one value different from the previous one.

4. **Model training**: after the pre-processing and the creation of the windows the USAD model is trained regularly, with the two phases process, as previously described (subsection 4.1.2).

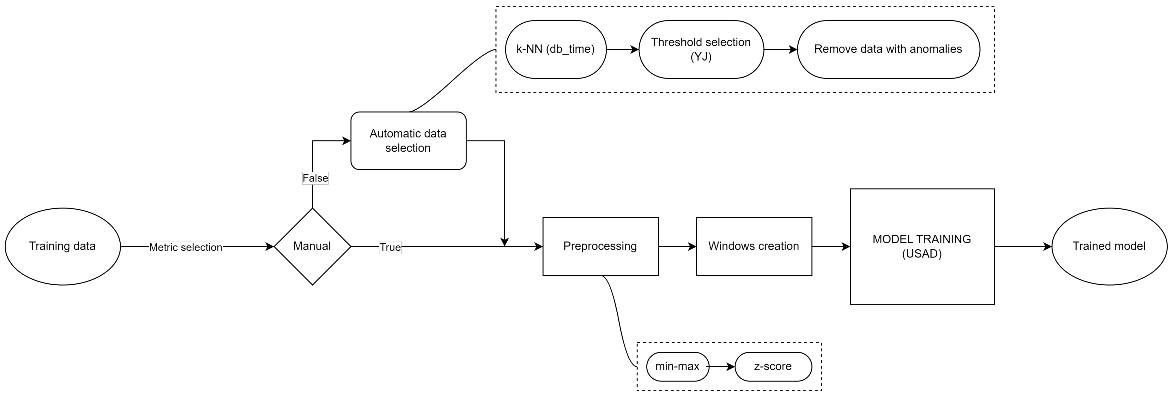


Figure 5.1: **Training** flow of the proposed solution

As for the inference of the proposed solution, the steps are the one represented in Figure 5.2:

1. **Data aggregation**: Since the USAD model is trained on data aggregated into window, the same step is needed for the inference. After the first w (window size) elements are collected, a prediction on the window is done, then when a new data point is collected the windows is slid by one and a new prediction is done.

2. **Model inference**: During this step the model computes an anomaly score for the windows passed as its input.

3. **Threshold selection**: Once the anomaly score is computed is necessary to fix a threshold in order to identify an anomalies from normal data points. Instead of opting for a manual selection of the threshold, approach not recommended since this value needs to be selected manually for each deployed model (one for each monitored database instance), an automatic approach is proposed, using the Inter-Quartile Region thresholder provided by the PyThresh library.

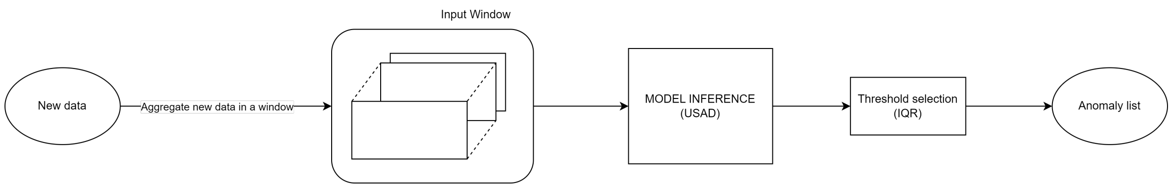


Figure 5.2: **Inference** flow of the proposed solution

5.2 Software Used

The entirety of this thesis, except for the data retrieval done in bash and SQL, is done using Python. This is justified by the fact that thanks to its simplicity and availability of machine learning libraries, Python has became the de facto standard for developing a machine learning application like the one proposed.

The main libraries used in the development of the code are:

- *Numpy*, *Pandas*.
- *PyTorch*: used for defining the machine learning model.
- *PyOD*: is an open-source library containing the implementation of over 40 anomaly detection algorithm (**?**); in this work is used for the k-NN algorithm (actually is a wrapper of the *scikit-learn* one).
- *PyThres*: a library containing more than 30 threshold algorithms (Perini et al. [2022]), used to set thresholds automatically, without the need to set a contamination level.
- *Plotly*: for plotting, mainly because it makes possible to export graphs into an *html* file, including also the javascript code tho make them interactive.

5.3 Automatic data selection

As explained in section 4.1, the USAD model uses an adversarial training framework to compensate the intrinsic problem of autoencoders model in the context of anomaly detection: that is the need to train only on “normal”data, to then learn to distinguish it from anomalous data points. This allows the USAD model to be more robust to noise in the training data, but will be proven in section 6.3 that choosing the training data with the less noise possible is beneficial in detecting anomalies consistently and with more precision.

Is also important to notice that the objective of the company is to deploy the application on multiple databases of a lot different clients, hence the need of a scalable solution. That is why the an automatic data selection module is provided.
The idea is to use the most important metric used for identifying problems in the database, the *DB time*, explained in detail in section 2.3. The k-NN clustering algorithm is used to compute the anomaly score of the data, but the problem in developing a solution that is independent from the data at hand (different machines will have different *DB time*, with different behavior) is the choosing of the threshold above which the anomaly score of a point will classify it as anomalous. This is usually done by selecting a percentage of expected outliers in the data, and the threshold will be set to a value that will make it so that the anomalies identified will be the percentage previously specified. This is of course a far from optimal solution, since is based on a esteem of the percentage of outliers in the data, and also it doesn’t take into account the intrinsic problem at hand, that is the different behavior of the databases, in this case is unreasonable to expect different databases to have the same percentage of anomalies.

This problem can be solved using the *Pythresh* library, that implements different methods to select the threshold without the need for hyper-parameters, allowing the data selection module to be independent from the monitored machine.
More precisely the algorithm used in the automatic data selection module is the Yeo-Jonson thresholder, that will be described in subsection 5.4.1.

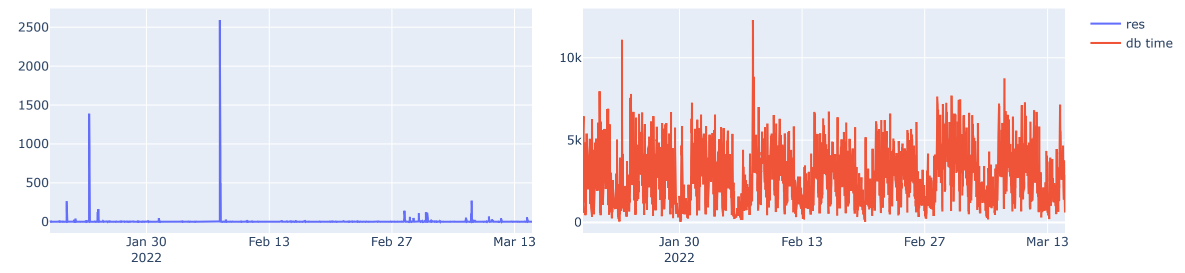
The automatic data selection module works as follows:

1. The k-NN algorithm is used to compute the anomaly score.
2. The Yeo-Jonson thresholder is used to compute the threshold.
3. The day or the week of data where the anomalies are found is discarded from the training set. Ideally it would be better to discard the entire week where anomalies are detected, but the choice depends on the quantity of the data available.

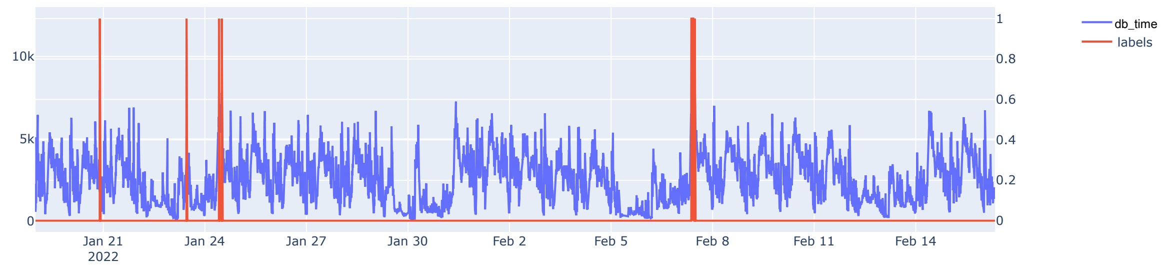
As for the threshold selection of the USAD model, this is done with the Inter-Quartile Region technique. The problem with this method is that the threshold is computed based on the distribution of the prediction score, hence the threshold will be a function of those scores; that is why a dynamic threshold selection method is proposed allowing for a continuous tuning of the threshold (subsection 5.4.3).

5.4 Threshold selection

As explained in the previous section, is fundamental to eliminate the need of hyper-parameters in the threshold selection step, since this allow to avoid the tuning of the thresholds to the specific database. The *PyThresh* library provides various thresolder



(a) *DB time* and anomaly score



(b) Identified anomalies

Figure 5.3: Automatic data selection example

selection algorithms, in the following subsections the ones used in this thesis work are described.

5.4.1 Yeo-Jonson transformation thresholder

This algorithm uses the Yeo-Johnson transformation (Yeo and Johnson [2000]) to evaluate a non-parametric means to threshold scores generated by the decision scores where outliers are set to any value beyond the max value in the YJ transformed data.
The Yeo-Johnson transformation is a power transform which is a set of power functions that apply a monotonic transformation to the dataset. For the decision scores this make their distribution more normal-like (Raymaekers and Rousseeuw [2021]).

$$\psi_{(y,\lambda)} = \begin{cases} \left((y+1)^{\lambda} - 1\right)/\lambda & \text{if } \lambda \neq 0,\, y \geq 0 \\ \log(y+1) & \text{if } \lambda = 0,\, y \geq 0 \\ -\left((-y+1)^{(2-\lambda)} - 1\right)/(2-\lambda) & \text{if } \lambda \neq 2,\, y < 0 \\ -\log(-y+1) & \text{if } \lambda = 2,\, y < 0 \end{cases}, \tag{5.1}$$

where λ is a power parameter that is chosen via maximum likelihood estimation, that is λ is the value that maximizes the log-likelihood function. Therefore, any values from the original decision scores that are beyond maximum value after this transformation are considered outliers. However, the closer a set of decision scores are to a normal distribution originally the smaller the probability this threshold will be able to identify outliers.

5.4.2 Inter-Quartile Region thresholder

Use the inter-quartile region to evaluate a non-parametric means to threshold scores generated by the decision scores where outliers are set to any value beyond the third quartile plus 1.5 times the inter-quartile region, as proposed in Bardet and Dimby [2017].
The inter-quartile region is given as:

$$IQR = |Q_3 - Q_1| \tag{5.2}$$

where Q_1 and Q_3 are the first and third quartile respectively.
The threshold for the decision scores is set as:

$$th = Q_3 + 1.5IQR \tag{5.3}$$

5.4.3 Dynamic Threshold

Selecting the threshold for the anomaly score produced by the USAD model has its own challenges, mainly due to the fact that a lot of historical data is needed to tune the threshold. This combined with the fact that the proposed solutions need "normal" data to learn a good representation of the distribution, poses a challenge in developing a solution easily deployable.

To solve this problem a non-parametric dynamic thresholding method is proposed. It is composed by the following steps:

1. **Initial threshold selection**: after the model is trained, the validation dataset is used to compute an anomaly score and then a threshold, with the Inter-Quartile Region technique

2. **At-Inference tuning**: once the model is deployed, the anomaly scores are collected and stored. Then the threshold is periodically re-computed using all the previously collected anomaly scores.

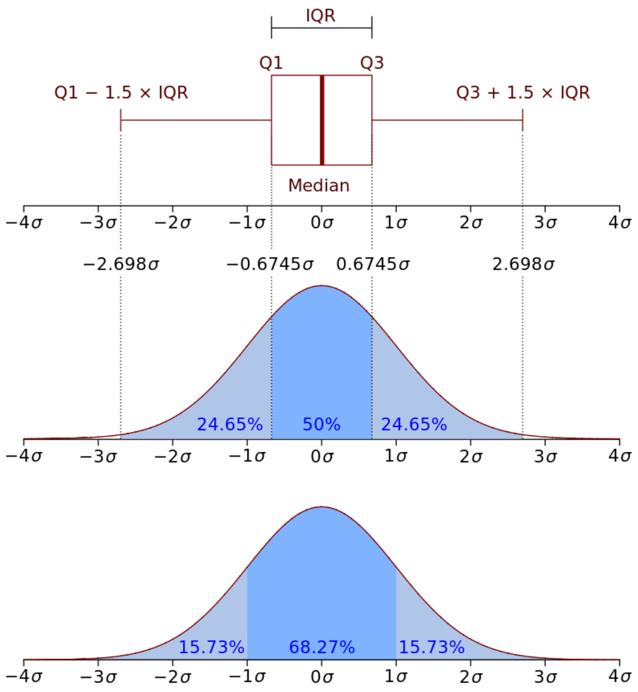


Figure 5.4: "Boxplot (with quartiles and an interquartile range) and a probability density function (pdf) of a normal $N(0, \sigma^2)$ population"Wikipedia [2023]

Given a reasonable amount of data used for training, (in the experiments done 3 weeks of clean data with mostly no anomalies was found sufficient) this method allows to have a solution that is deployable with approximately a month of collected data and preferably one or two weeks for tuning.

The problem that remains to be solved by this solution is that some anomalies have to be detected to do a correct tuning of the threshold, if not the model will be more sensitive to outlier at the beginning. Thankfully since the α and β parameters of the USAD model can be modified during inference, they can be used for compensating this behavior until the threshold is correctly tuned.

5.5 Hyperparameter Selection

Even after removing the critical parameter needed to select the threshold, however, the proposed solution still has many parameters, some related to the USAD model, others to the application in general, that need tuning; a complete list of the parameters is provided in Table 5.1.

Of particular importance are the following hyper-parameters:

- *downsampling_rate*: This parameter allows to aggregate and average the original data. This is useful since it will noticeably speed-up the training of the model. In the specific use-case for the company, having a data granularity of one minute wasn't necessary and the data available was enough even after the downsampling, hence the decision of aggregating the data each 5 minutes.

- *window_size*: This is the parameter w of the USAD model (section 4.1), representing the number of data points used to generate the windows. In this case, thanks to the sector expert's opinion, the choice of using a window size of 12 (creating windows of 60 minutes) was taken, since USAD can detect behavior changes faster when the window size is smaller because each observation has a greater impact on the anomaly score. A larger window will detect longer anomalies, but it will have to wait for more observations to detect an anomaly. However, if an anomaly is too short, it may be hidden in the number of points that a too-large window has.
- *metrics*: This parameter allows the user to specify the list of metrics to analyze. By default the metric used are the one selected and discussed in section 2.3.
- α *and* β: This are two parameter of the USAD model; they must respect the condition $\alpha + \beta = 1$ and their function is to parameterize the trade-off between true positives and false positives. Increasing α will decrease the number of false positives (*low sensitivity scenario*), while increasing β will have the opposite effect, raising the number of false positives (*high sensitivity scenario*). This parameters can be changed on-the-fly at runtime, giving the operators the possibility to tune the sensitivity of the algorithm based on their workload.

Parameter	**Category**	**Description**	**Default**
multivariate	general	if the analysis is conducted on multiple metric using the USAD model or on a single metric using the k-NN algorithm	*True*
th_algorithm	threshold	the thresholding algorithm to use	iq yj
contamination		the data expected contamination rate, used if no thresholding algorithm is used	*None*
auto_ds	data selection	use the automatic data selection module	*True*
training_start		used if the data is not automatically selected	*None*
training_end			
downsampling	preprocessing	number of data point to aggregate in each sample	5
window_size		number of data point used to create the window	12
normalization		normalization technique or techniques used	*z-score, min-max*
metrics		list of metrics to analyze	
batch_size	USAD	sizes of batches used in training	32
epochs		number of training epochs	100
hidden_size		hidden size of the USAD model	10
α	inference	increasing α will decrease the number of FP	0.5
β		increasing β will raise the number of FP	0.5

Table 5.1: This table contains the main hyperparameters of the application, grouped by category. The last column contains the best values tuned for the specific dataset considered in this thesis and tuned to meet the company needs.

Chapter 6

Results

During the writing of this thesis extensive tests were conducted, in order to understand the potential of different models and identify a suitable solution, for finding the best thresholding algorithm and generally for tuning the various hyperparameters. Once the final solution was developed, the presented following 4 experiments were considered the most interesting and useful to test the goodness of the proposed solution.

This chapter will be organized as follows:

1. at first an introduction on the data used to train and test the model will be given
2. then two test set will be presented, one containing data with fewer anomalies but generated in chronological order after the data using for testing, and it will be used to simulate a real application of the model; meanwhile the second set of data point contains statistics collected before the training data, but it will contain two big anomalies that are considered essential to be recognized by the application in order to be of any use.
3. finally four experiment exploring the different capabilities the proposed solution has to offer will be presented

6.1 General setting

The data used for the development of the application belongs to a client of the company, and was collected between the end of January and the beginning of March 2022. Data of two database instances was collected, but the main focus is placed on the main instance for which more data is available and especially because there is a period with no anomalies, at least in the *DB TIME*; this does not mean there weren't anomalies, but at least the database has not slowed down.

Figure 6.1 shows the *DB TIME* of the two instances. As explained in section 2.3, this metric will be used as a first parameter to analyze the results of the model, and the approach, as suggested by the company experts, will be to consider a test successful if the anomalies in highlighted in red in the figure are detected. This is a far from optimal solution to completely evaluate the model, activity for which a thorough analysis of the company experts is mandatory, but all expert consulted agreed in using this approach as a first screening of the results, for then proceed to a more detailed analysis.

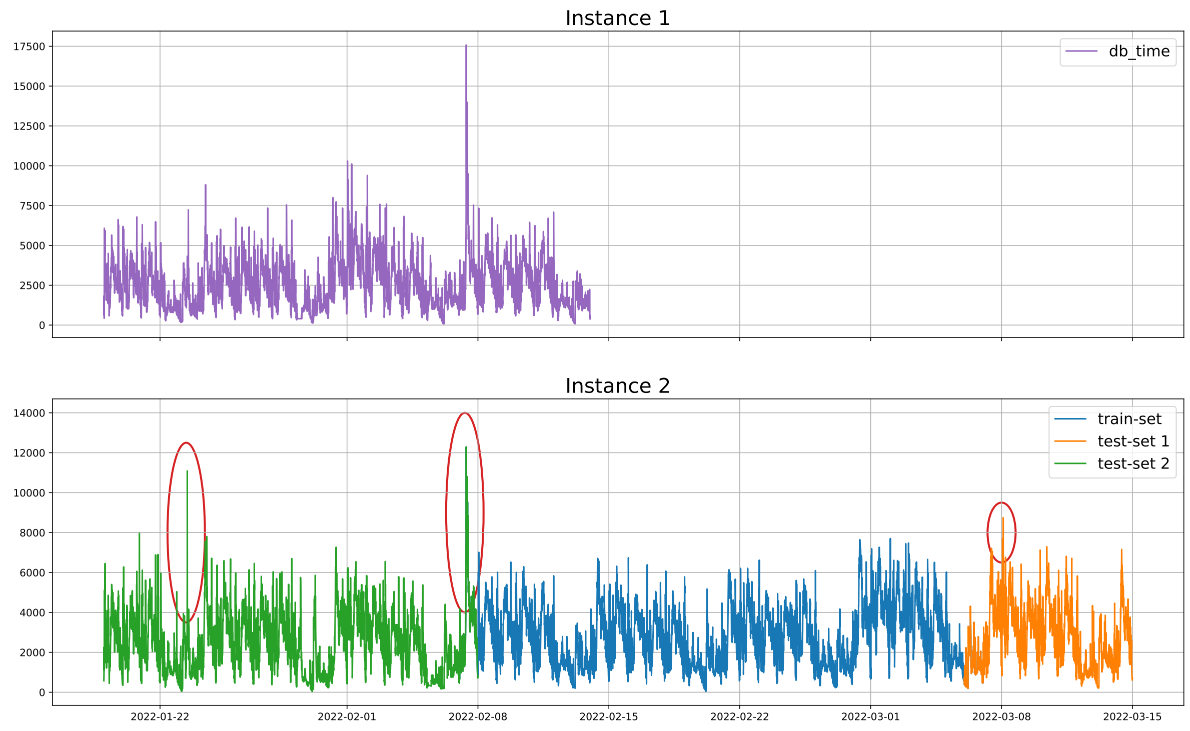


Figure 6.1: Training and test data from 2 databases instances (for simplicity only the *DB TIME* metric is shown since it its the most important metric in detecting anomalies but the actual data contains information of 155 metrics)

As previously explained, the autoencoder-based model adopted, has the intrinsic limitation of needing a training dataset with the less possible amount of anomalies. That is why the four weeks of data, from 2022-02-07 to 2022-03-05 (Figure 6.2), of the main database instance, is used in all experiments (except from the test on the automatic training data selection) for training, since no spikes in *DB TIME* are present.

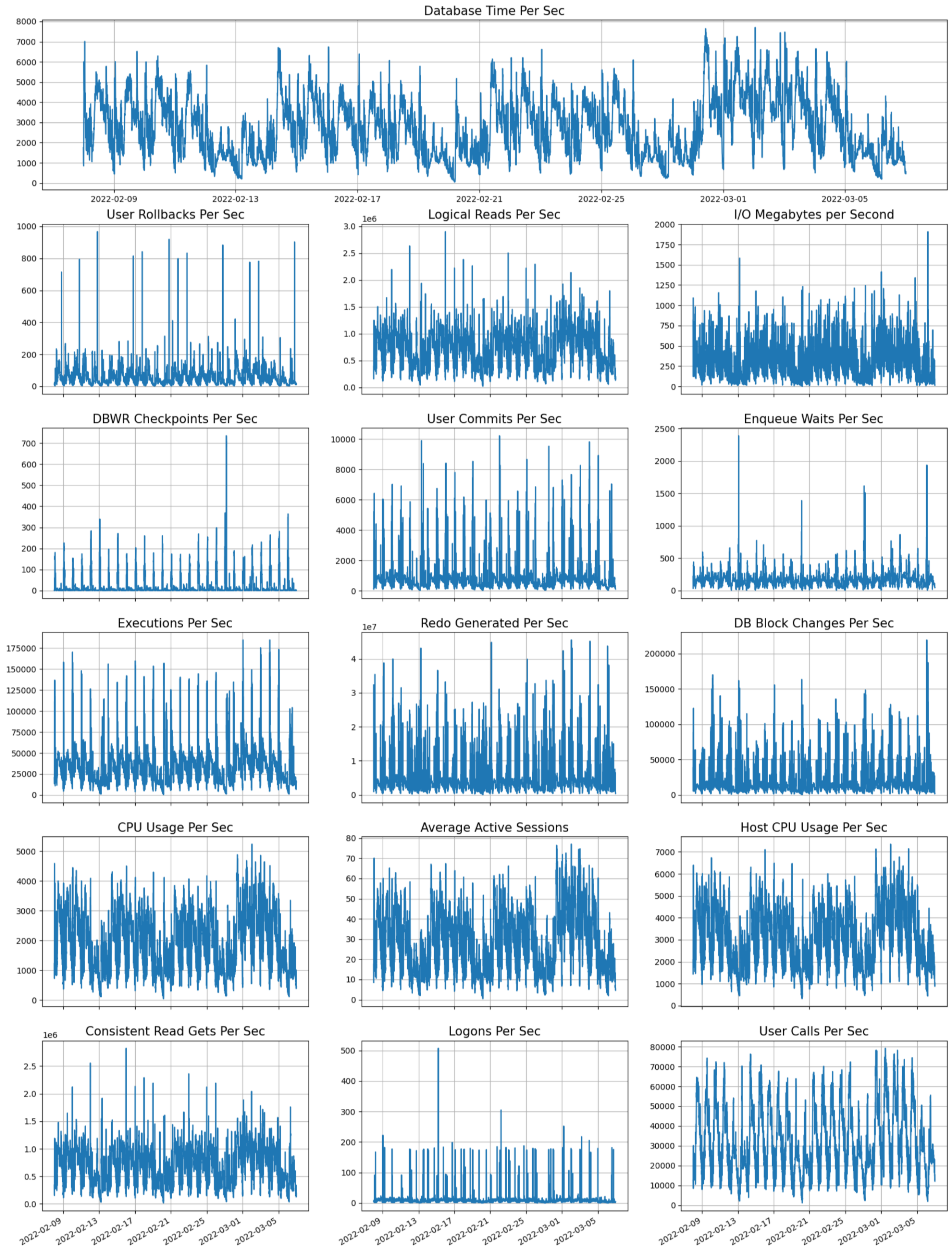


Figure 6.2: Metrics used in the experiments for training the USAD model

For what concerns the test data, two sets are used, as described in the next subsections.

- The first test set consists of the values immediately consecutive to those used during training, in order to simulate a real scenario of immediate use of the application. This is a particularly difficult case to evaluate, since there are no major anomalies to detect.

- The second test set consists of data collected previously of the one used for training. Differently from the first one, this dataset contains two main anomalies, one is a sudden spike of less than 5 minutes in the *DB TIME*, that can be considered as an anomaly, but due to its short duration can be overlooked; the second one instead is more prolonged, for roughly four ours. This last anomaly ose considered mandatory to detect, if not the test is failed.

6.1.1 Hyperparameters

In all the following experiments the hyperparameters used are the same of Table 5.1, in order to focus only on the effect of the changes of data used for training and testing of the model. Is important to notice that fine-tuning the hyperparameters can improve the performance of the model, but the ones used allow for the best results overall.

The effect of the parameters α and β is not discussed, since their usage is meant for tuning the sensitivity on the model at runtime, mainly to change the number of anomalies signaled by the application, based on the number of operators and the amount of work of the company at any given time.

6.2 Experiment 1: Manual training data selection

In this first experiment, the data for training is manually selected, that is the data used is the one of Figure 6.2.

6.2.1 Test data 1

As shown in Figures 6.3, 1 and 2, two anomalies are detected in this period, the first one for 1.5 hours and the other during a period of time of 45 minutes. The first one is the *DB TIME* anomaly also highlighted in Figure 6.1, and looking at all the metrics in that time frame is imputable to the spikes in *Executions Per Sec* and *Average Active Sessions*. The second one is more interesting since it is not in proximity of a clear peak in the *DB TIME*, thus the users probably didn't notice any slowdowns in the database, but it that period the *CPU Usage Per Sec* reached is maximum value and the *Host CPU Usage Per Sec* has a spike in its state. In the same period the *Average Active Sessions* and *User Calls Per Sec* reached a global maximum, indicating a possible period of stress for the database.

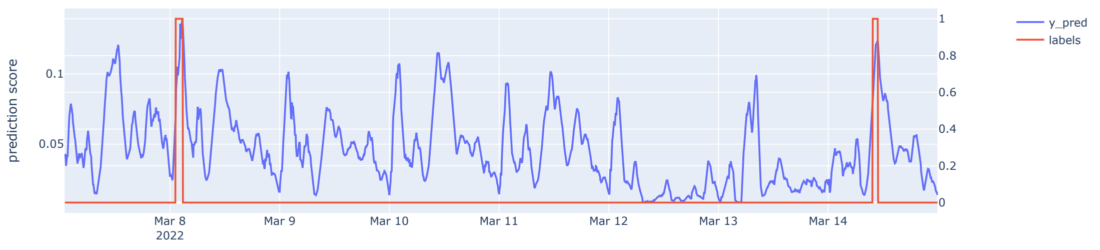


Figure 6.3: Anomaly score of the model, in red the values selected as anomalies

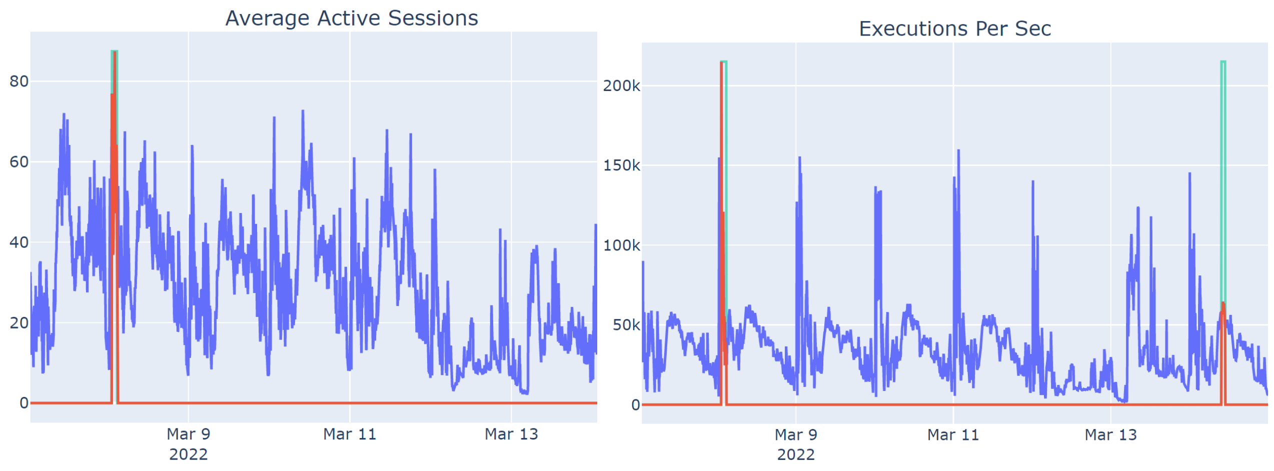


Figure 6.4: Anomalies experiment 1.1

6.2.2 Test data 2

In this second case 3, 4, the first anomaly on the *DB TIME* is not detected correctly, but other than the already mentioned short duration of the anomaly making very difficult to detect. The fact that this spike occurs early on in the detection process hence the threshold is probably not completely tuned is to exclude since as shown in the anomaly scores of Figure 6.5, there is no peaks in the morning of the 01-24-2022. The first anomaly detected in this period is not visible looking only at the *DB TIME*, but the model found an anomaly in correspondence of a span of time of 3 hours, where the *Consistent Read Gets Per Sec*, *Physical Reads Per Sec*, *Logical Reads Per Sec*, *I/O Megabytes per Second*, *User Commits Per Sec* and *DB Block Changes Per Sec* all had an anomalous behavior. The last anomaly detected is in the same period where the biggest peak of the *DB TIME* is reached, probably due to the abnormal increase in the write operations (*Physical Writes Per Sec*, *I/O Megabytes per Second*) and the *Average Active Sessions*.

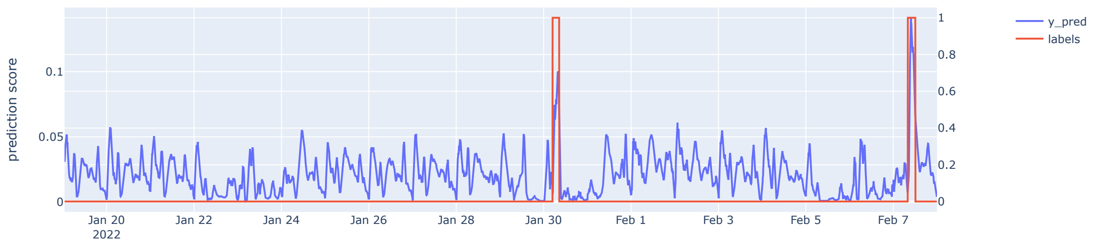


Figure 6.5: Anomaly score of the model, in red the values selected as anomalies

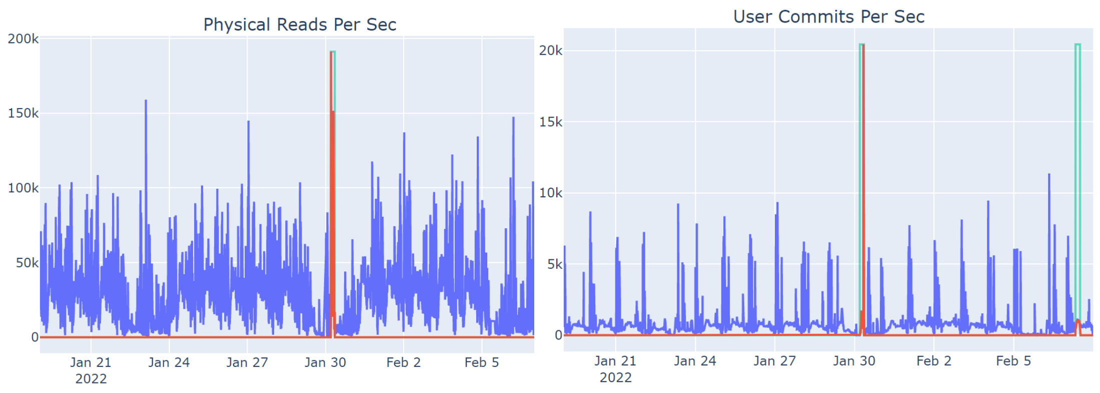


Figure 6.6: Anomalies experiment 1.2

6.3 Experiment 2: Automatic training data selection

In this experiment the automatic training data selection module developed is tested.

The *DB TIME* data collected for 7 weeks is passed to the univariate timeseries analysis module to detect anomalies on all the data available; subsequently the data with no anomalies is used for training and the remaining data is used in conjunction with the validation set to tune the threshold of the USAD model; then the multivariate analysis is conducted.

In Figure 6.7 the anomaly detected by the univariate module, that uses the k-NN algorithm to compute the anomaly score and the Yeo-Jonson thresholder to identify the anomalies, are shown. The data from the 7 to the 27 of February 2022 is selected as optimal for training.

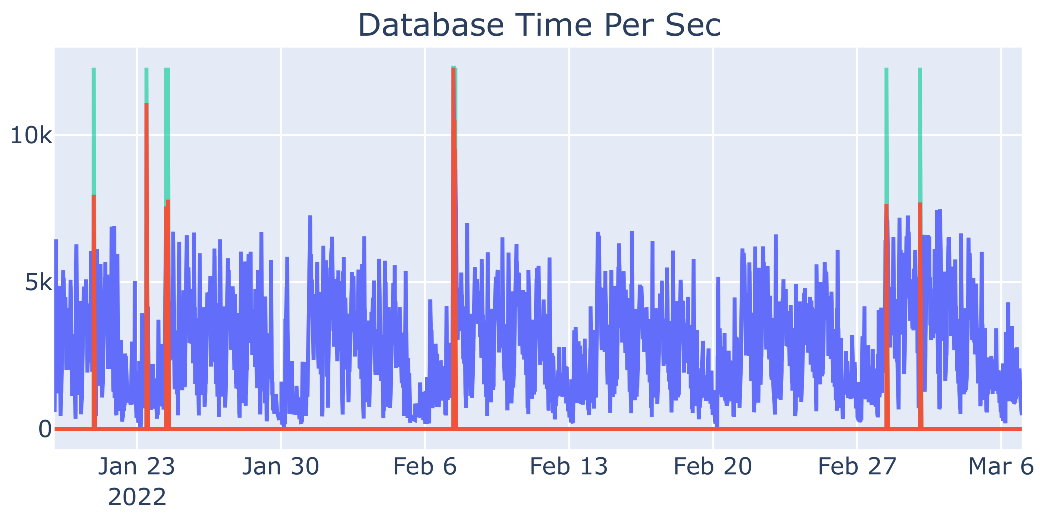


Figure 6.7: *DB TIME* detected by the univariate model

The mulitvariate analysis conducted by the trained (Figures 5, 6) model correctly identify only one anomaly of 2 hours on the 03-08-2022 night. Compared to the model trained with the data selected manually (Figures 1, 2) the first evident anomaly is correctly identified by both models, but the model analyzed in this experiment doesn't identify the second presumed anomaly identified in the first experiment. As discussed in the precedent section, this point is considered, even by the company's experts, a difficult point to label; with the data of the 19 metrics used by the model the experts consulted agreed on not considering it an anomaly, since they tend to prefer a non too sensitive model, that identify only the anomalies that require immediate actions.
Comparing the anomaly score of the two models (Figure 6.3 for the model of Experiment 1 and Figure 6.8 for this experiment) we can see that both of them have an high anomaly score in presence of the anomaly in question, but the better tuned threshold of the second

model allows for a more precise distinction, identifying only the correct outlier.

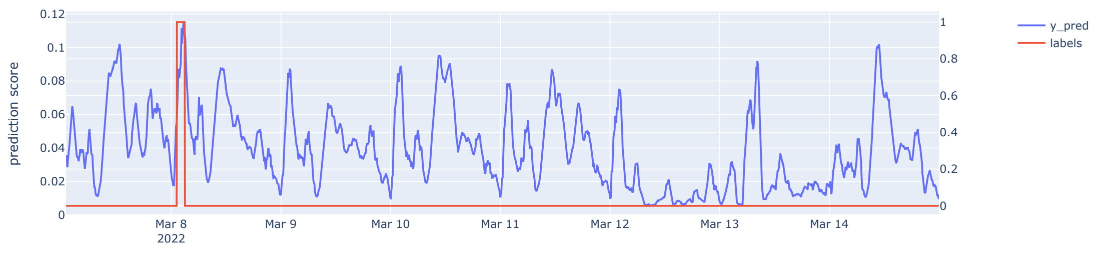


Figure 6.8: Anomaly score of the multivariate model, in red the values selected as anomalies

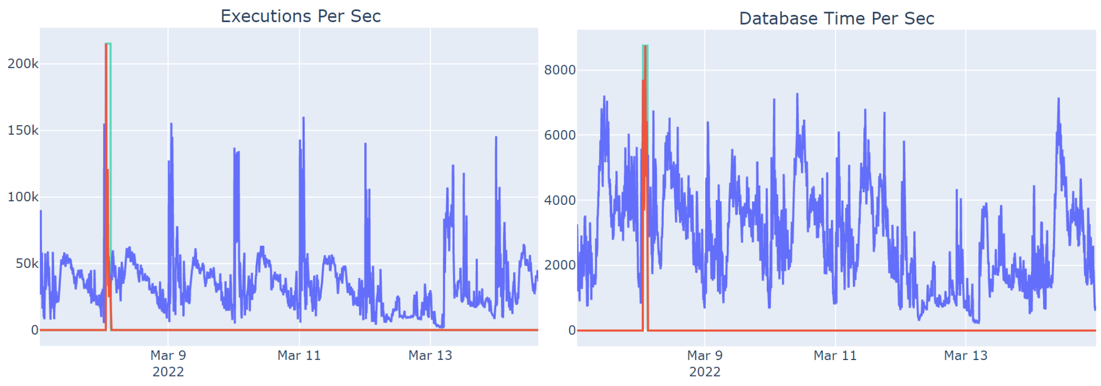


Figure 6.9: Anomalies experiment 2.1

6.4 Experiment 3: Test on large number of metrics

The input data used for this project is the one collected in the *V$SYSMETRIC* view available in the Oracle Database environment. Of the 155 metrics collected in this table, to simplify the task of the model, only 19 were selected as they are considered the most representative of the database health status by the company experts. In this experiment, a larger set of metrics is considered, in order to see the effects of increasing the number of metrics and analyze the consequences of doing so. For this experiment, a set of 60 metrics is selected, containing statistics sampled both per second and per transaction; a transaction is a logical, atomic unit of work that contains one or more SQL statements.

A transaction groups SQL statements so that they are either all committed, which means they are applied to the database, or all rolled back, which means they are undone from the database.

In both cases with the two different test datasets, an increase in the detected anomalies can be noticed due to the anomalous behavior of the new metrics considered, but the model seems to scale well, since the anomaly detected in Experiment 1 are still found by the algorithm, thus indicating that increasing the number of metrics doesn't affect the ability of the model to detect anomalies, but it simply finds other outliers because of the newly added metrics, that have a trend the differs from the normal one on different times compared to the metrics considered in Experiment 1.

However, it is important to underline the fact that using many metrics has two main disadvantages:

1. the first is that both training and inference take significantly longer times than the case with fewer metrics.

2. the second is that many metrics among those added may have anomalous behavior but that anomalies in their values very often do not lead to slowdowns in the database and therefore using them would only lead to an increase in reports that operators would then have to analyze.

However, the use of a high number of metrics can be useful in the case of a non-real time analysis, in which anomalies often not taken into consideration by the operators can be useful in discovering the causes of any database performance problems.

6.4.1 Test data 1

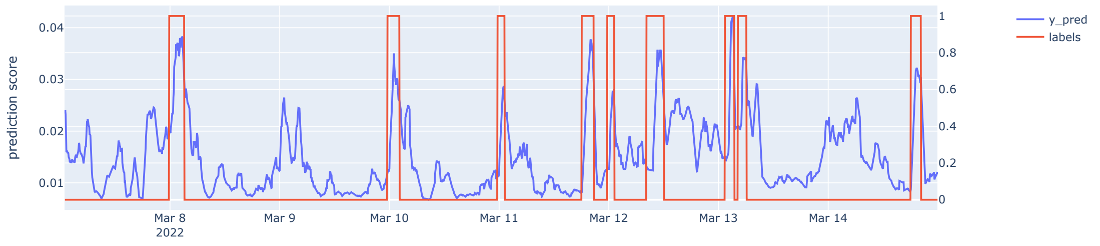


Figure 6.10: Anomaly score of the multivariate model, in red the values selected as anomalies

6.4.2 Test data 2

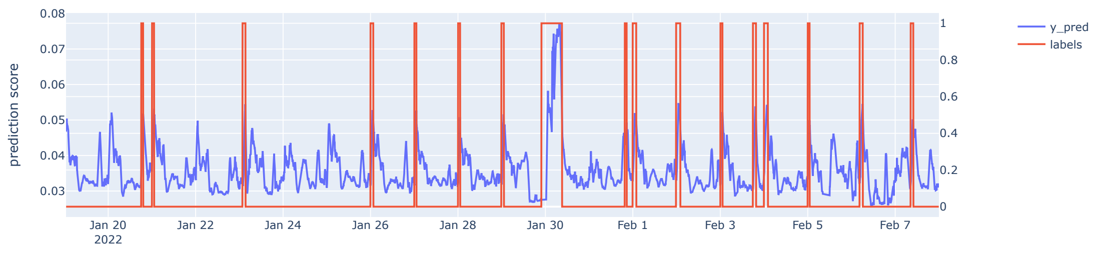


Figure 6.11: Anomaly score of the multivariate model, in red the values selected as anomalies

6.5 Experiment 4: Test on data from a different database instance

In order to test the generalization capabilities of the proposed solution, for this experiment, the trained model of Experiment 1 is tested with data from a different database instance running on a different server in the client cluster (RAC). However is important to specify that this test on a different database is possible only if the customer's architecture is well structured, i.e. that the machines are configured in the same way, both in terms of hardware and software.

In this case we show that the model performs very well, thus proving that is possible to train only one model, and then using it to detect anomalies on multiple database instances, reducing the cost of training, and if the prediction frequency is not to small, it is even possible to use only one model for the inference, reducing significantly the computational cost of the proposed anomaly detection application.

In Figures 19 and 20 the results of this test are presented. The second and fourth anomalies detected are in concomitance of 2 spikes in the *DB TIME*, *Physical Reads Per Sec* and *Logons Per Sec*. The first and fifth anomalies detected (less that one hour) have not a noticeable impact on the *DB TIME*, but are caused by the *Physical Reads Per Sec* and *Physical Writes Per Sec* for the first one and by the *Average Active Sessions* for the other one. As for the third anomaly, only lasted an hour, is not clear why the model identified this period as anomalous, and after an in-depth analysis the conclusion that this anomaly is wrongly signaled as such is reached.

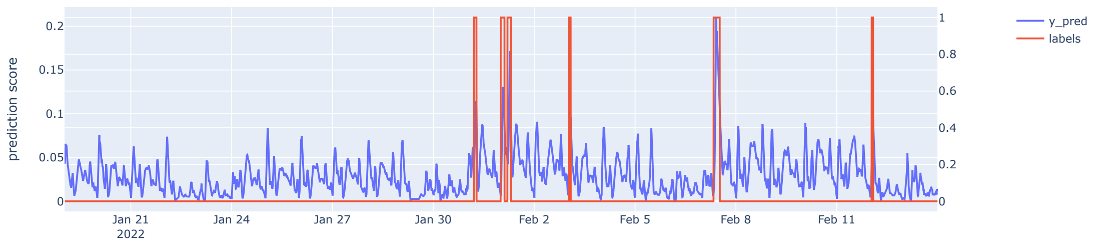


Figure 6.12: Anomaly score of the multivariate model in a different database instance, in red the values selected as anomalies (experiment 4)

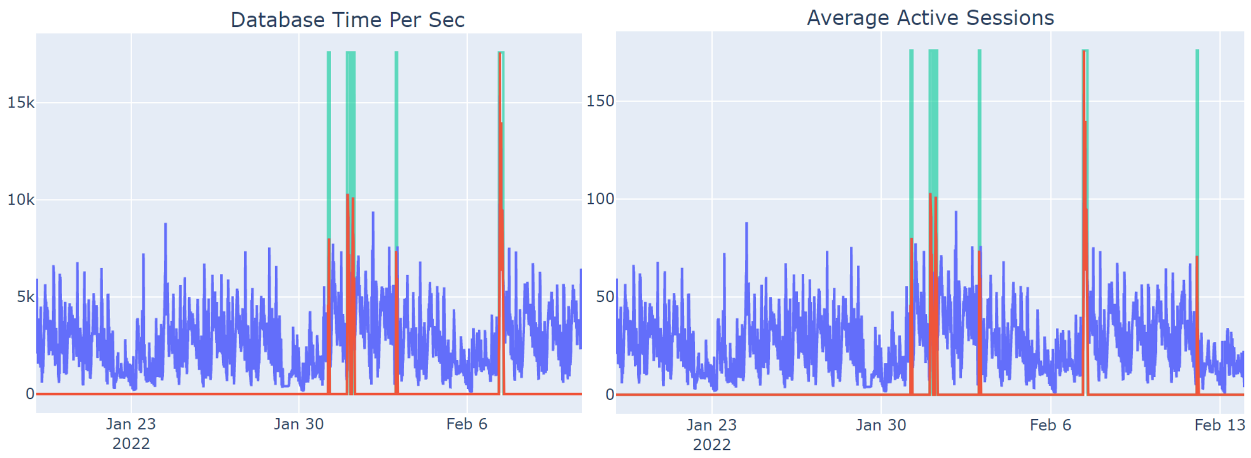


Figure 6.13: Anomalies experiment 4

Chapter 7

Conclusions

This thesis presented an unsupervised anomaly detection method for multivariate time series in an Oracle database, using adversarially trained autoencoders. The main research question was how to effectively detect anomalies in an Oracle database without requiring labeled data or manual selection of training data.

The main findings of this study were:

- The proposed method can automatically select normal data from the database metrics to train the autoencoders, and use dynamic thresholds to identify anomalies in new data.
- The proposed method can generalize to different Oracle database instances, and provide useful insights for database performance monitoring.
- The general and scalable architecture of the application developed is a useful asset to the company in where it was developed, offering an already production ready and documented framework to implement other anomaly detection methods in the future and useful tools for visualizing in an interactive way the results of the analysis.

The main contribution of this study were:

- The implementation of a state-of-the-art unsupervised anomaly detection method for multivariate time series in an Oracle database, based on adversarially trained autoencoders, in a production context.
- A general and scalable architecture for implementing the method, using Python and PyTorch.
- A comprehensive evaluation of the method on real-world data, collected from different Oracle database instances.
- A practical application of the method for database performance monitoring, providing a tool for database administrators to identify potential problems and optimize their systems.

The main limitations of this study were:

- The lack of ground truth labels for the anomalies, which made it difficult to validate the results objectively and quantify the performance of the method.
- The dependence of the method on the quality and quantity of the data available from the Oracle database, which might vary depending on the database configuration and usage.
- The assumption that only normal data is used for training, which might not hold in some scenarios where anomalies are frequent or unavoidable.

7.1 Future works

This study has demonstrated that adversarially trained autoencoders can be a powerful technique for unsupervised anomaly detection in multivariate time series in an Oracle Database. By automatically selecting normal data from the database metrics, using dynamic thresholds to identify anomalies, and generalizing to different Oracle database instances, the proposed method can provide a valuable tool for database performance monitoring. This study has also contributed to the existing literature on anomaly detection by applying a novel method to a real-world problem. However, there is still room for improvement and further research in this field. Therefore, this study hopes to inspire other students and researchers to explore new ways of detecting anomalies in complex systems using machine learning techniques.

The main recommendations for future research are:

- To explore other techniques for selecting normal data from the database metrics, such as clustering or outlier detection methods.
- To investigate other ways of setting dynamic thresholds for anomaly detection, especially their tuning over time.
- To extend the method to other types of databases or time series data, such as Microsoft SQL Server.
- To incorporate feedback from domain experts or users to improve the interpretability and usability of the method.
- Implementing a rest API to send notification to the performance monitoring panel, used by the company, when an anomaly is detected.

Bibliography

[1] Fabrizio Angiulli and Clara Pizzuti. Fast outlier detection in high dimensional spaces. In Tapio Elomaa, Heikki Mannila, and Hannu Toivonen, editors, *Principles of Data Mining and Knowledge Discovery*, pages 15–27, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-45681-0.

[2] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A. Zuluaga. Usad: Unsupervised anomaly detection on multivariate time series. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3395–3404, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403392. URL https://doi.org/10.1145/3394486.3403392.

[3] Pierre Baldi. Autoencoders, unsupervised learning and deep architectures. In *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27*, UTLW'11, page 37–50. JMLR.org, 2011.

[4] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders, 2021.

[5] Jean-Marc Bardet and Solohaja-Faniaha Dimby. A new non-parametric detector of univariate outliers for distributions with unbounded support, 2017.

[6] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, sep 1975. ISSN 0001-0782. doi: 10.1145/361002.361007. URL https://doi.org/10.1145/361002.361007.

[7] Ane Blázquez-García, Angel Conde, Usue Mori, and Jose A. Lozano. A review on outlier/anomaly detection in time series data, 2020.

[8] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), jul 2009. ISSN 0360-0300. doi: 10.1145/1541880.1541882. URL https://doi.org/10.1145/1541880.1541882.

[9] Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. Mode regularized generative adversarial networks, 2017.

[10] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative adversarial networks: An overview. *IEEE*

Signal Processing Magazine, 35(1):53–65, jan 2018. doi: 10.1109/msp.2017.2765202. URL https://doi.org/10.1109%2Fmsp.2017.2765202.

[11] Arden Dertat. towardsdatascience.com, 2017. URL https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798.

[12] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[13] Douglas M Hawkins. *Identification of outliers*, volume 11. Springer, 1980.

[14] G.S. Maddala. *Introduction to Econometrics*. Wiley, 2001. ISBN 9780471497288. URL https://books.google.it/books?id=BBa3AAAAIAAJ.

[15] Stephen M Omohundro. *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.

[16] oracle world.com. Logical storage structures, 2023. URL https://www.oracle-world.com/dba-basic/logical-storage-structures/.

[17] oracletutorial.com. Oracle database architecture, 2023. URL https://www.oracletutorial.com/oracle-administration/oracle-database-architecture/.

[18] Lorenzo Perini, Paul Buerkner, and Arto Klami. Estimating the contamination factor's distribution in unsupervised anomaly detection, 2022.

[19] Elad Plaut. From principal subspaces to principal components with linear autoencoders, 2018.

[20] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. *SIGMOD Rec.*, 29(2):427–438, may 2000. ISSN 0163-5808. doi: 10.1145/335191.335437. URL https://doi.org/10.1145/335191.335437.

[21] Jakob Raymaekers and Peter J. Rousseeuw. Transforming variables to central normality. *Machine Learning*, mar 2021. doi: 10.1007/s10994-021-05960-5. URL https://doi.org/10.1007%2Fs10994-021-05960-5.

[22] David Vint, Matthew Anderson, Yuhao Yang, Christos Ilioudis, Gaetano Di Caterina, and Carmine Clemente. Automatic target recognition for low resolution foliage penetrating sar images using cnns and gans. *Remote Sensing*, 13:596, 02 2021. doi: 10.3390/rs13040596.

[23] Wikipedia. K-nearest neighbors algorithm — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=K-nearest%20neighbors%20algorithm&oldid=1150518096, 2023. [Online; accessed 27-April-2023].

[24] Wikipedia. Quartile — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Quartile&oldid=1141261931, 2023. [Online; accessed 27-April-2023].

[25] In-Kwon Yeo and Richard A. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 2000. ISSN 00063444. URL http://www.jstor.org/stable/2673623.

Appendix

In this appendix the detailed results of the application, especially showing the behavior of each metric in detail, are provided.

Experiment 1: Manual training data selection

A. Test data 1

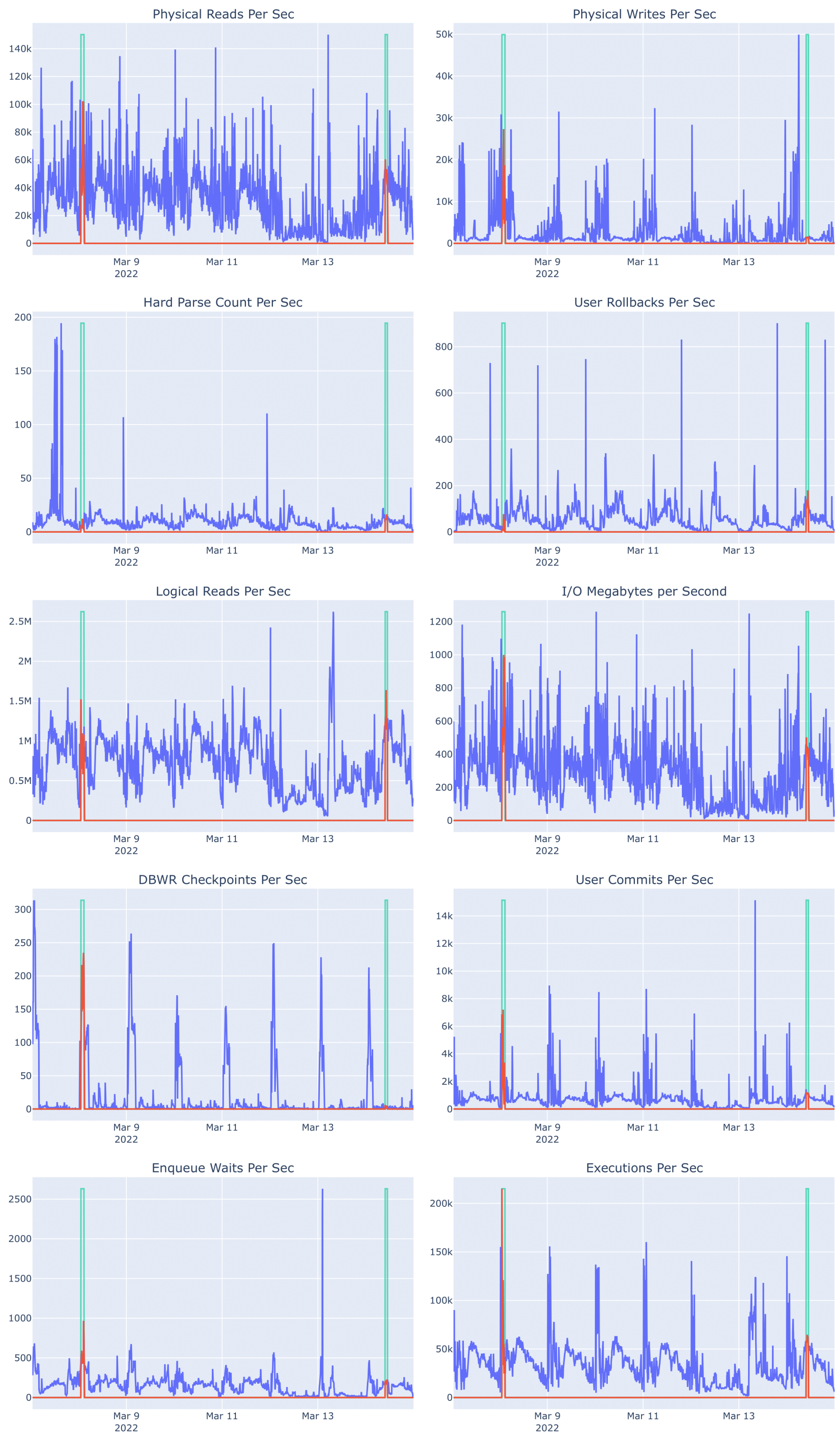


Figure 1: Metrics (Part 1) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.

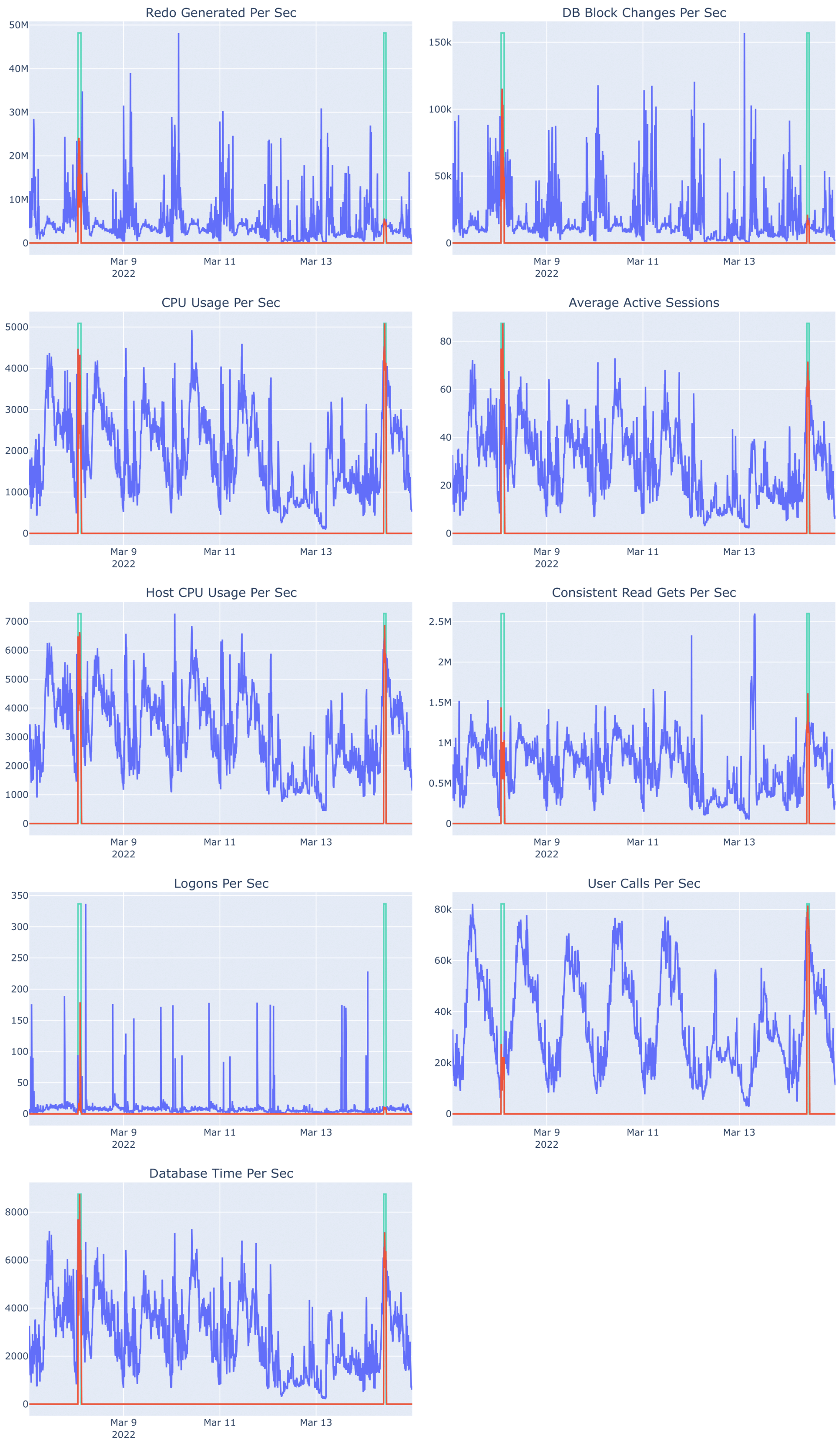


Figure 2: Metrics (Part 2) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.

B. Test data 2

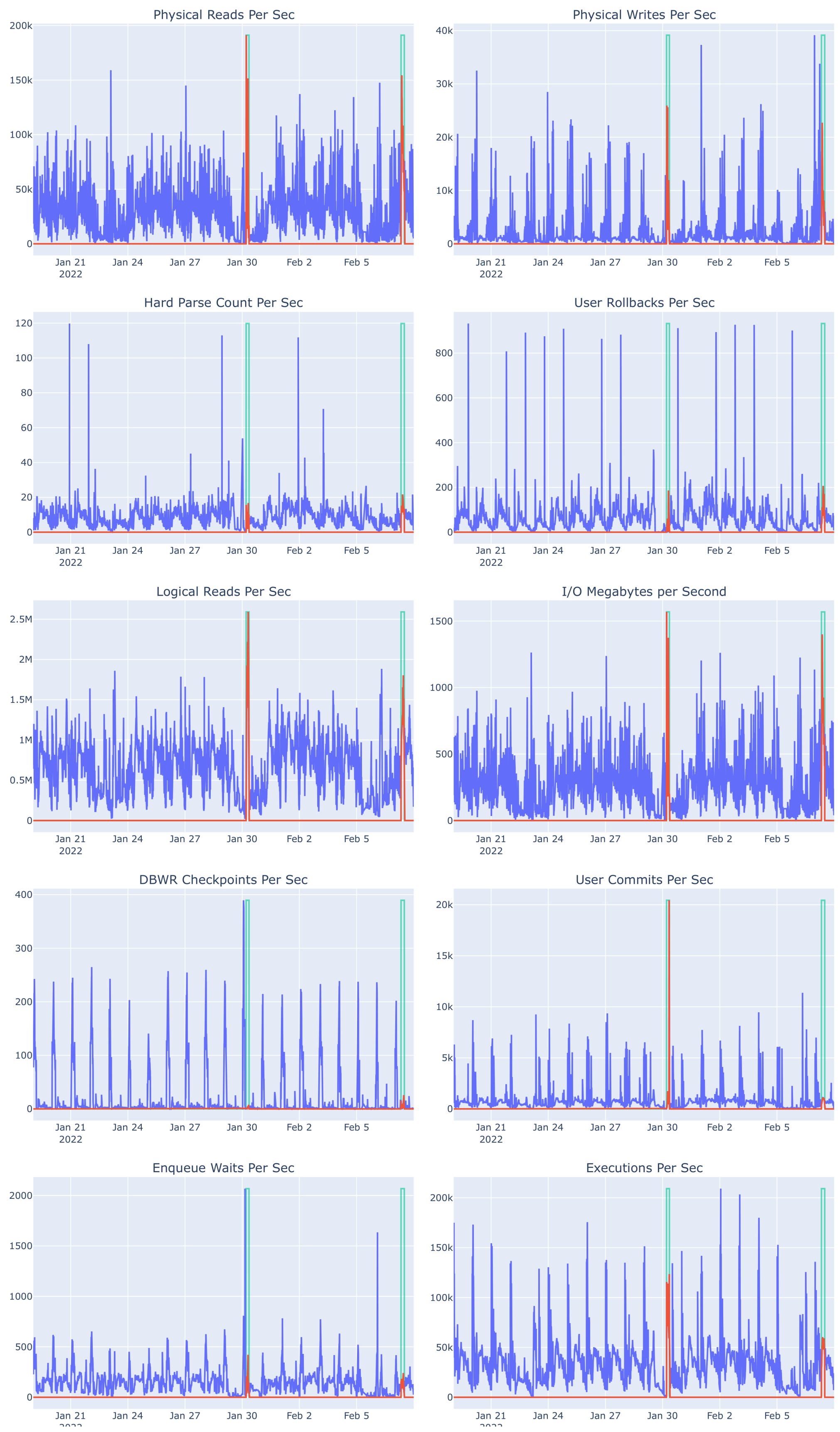


Figure 3: Metrics (Part 1) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.

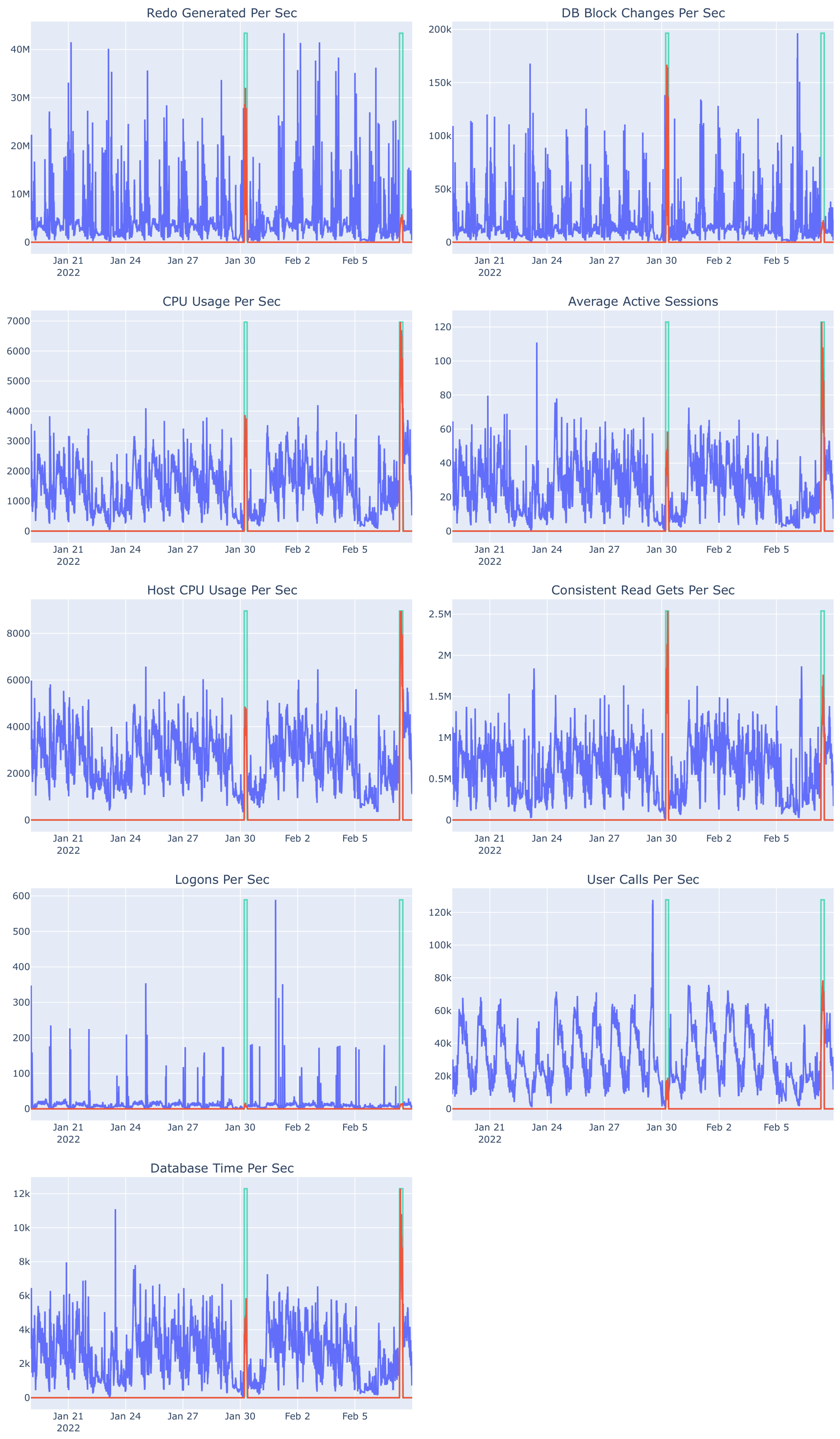


Figure 4: Metrics (Part 2) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.

Experiment 2:
Automatic training data selection

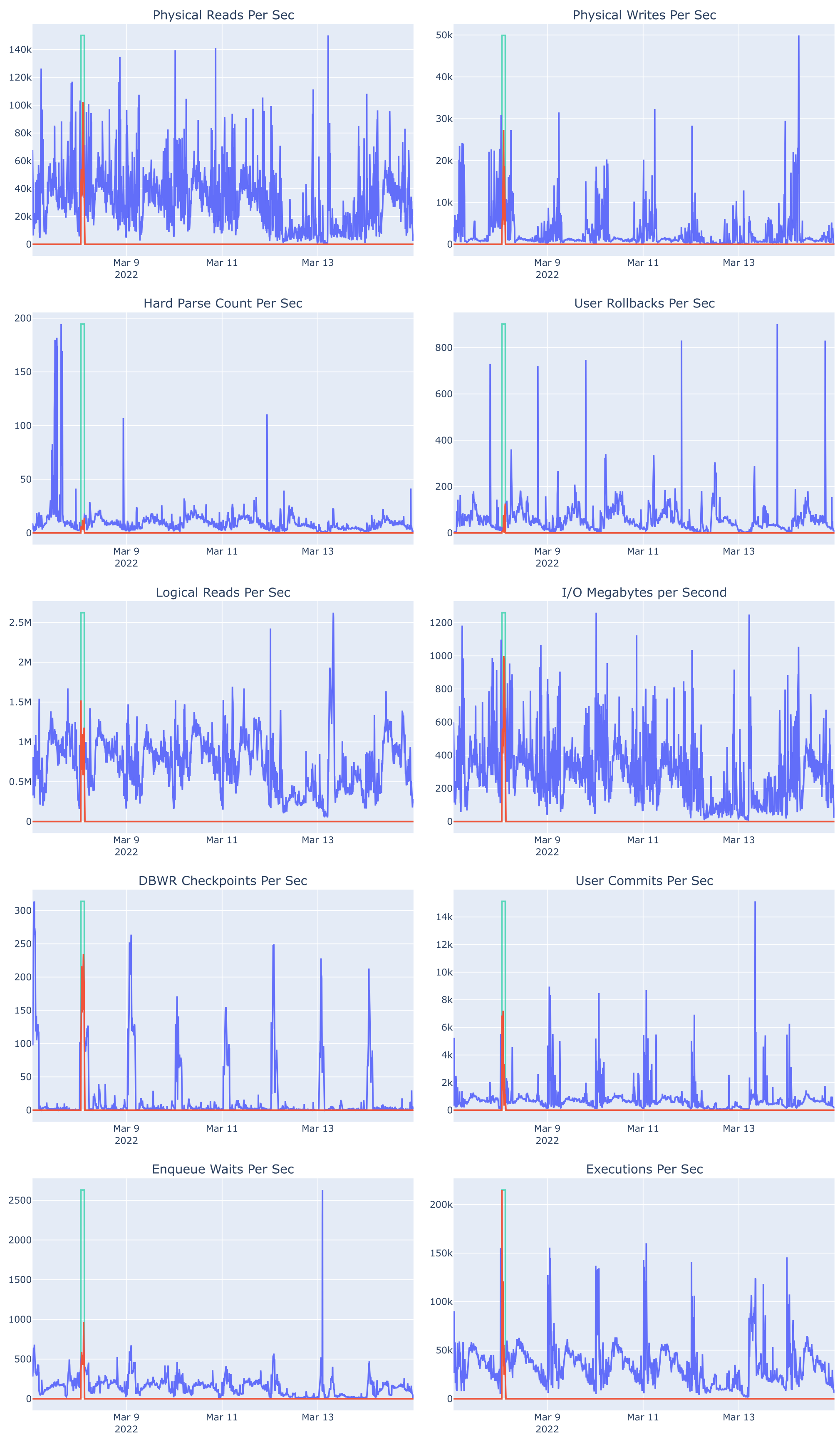


Figure 5: Metrics (Part 1) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.

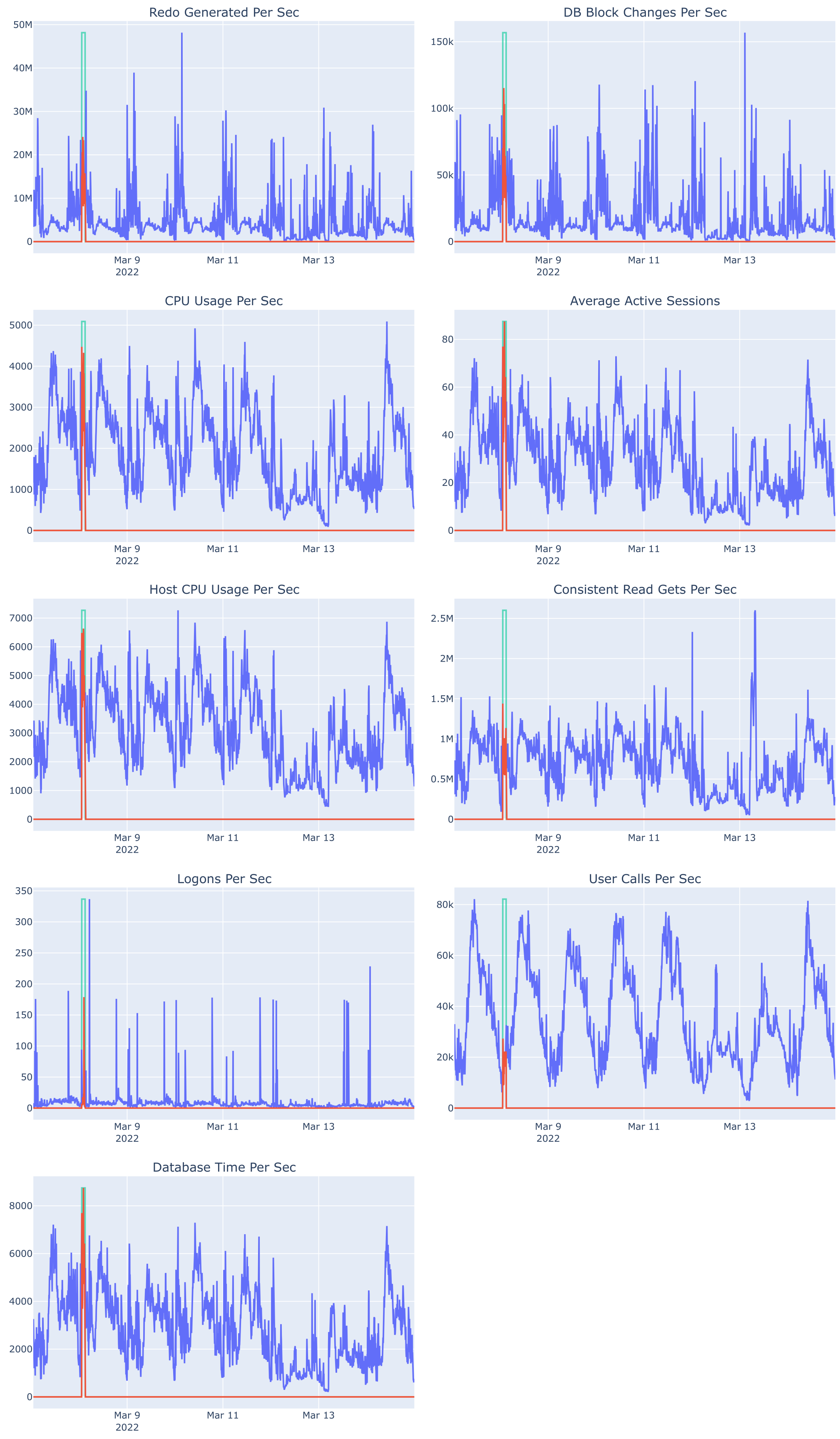


Figure 6: Metrics (Part 2) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.

Experiment 3:
Test on large number of metrics

A. Test data 1

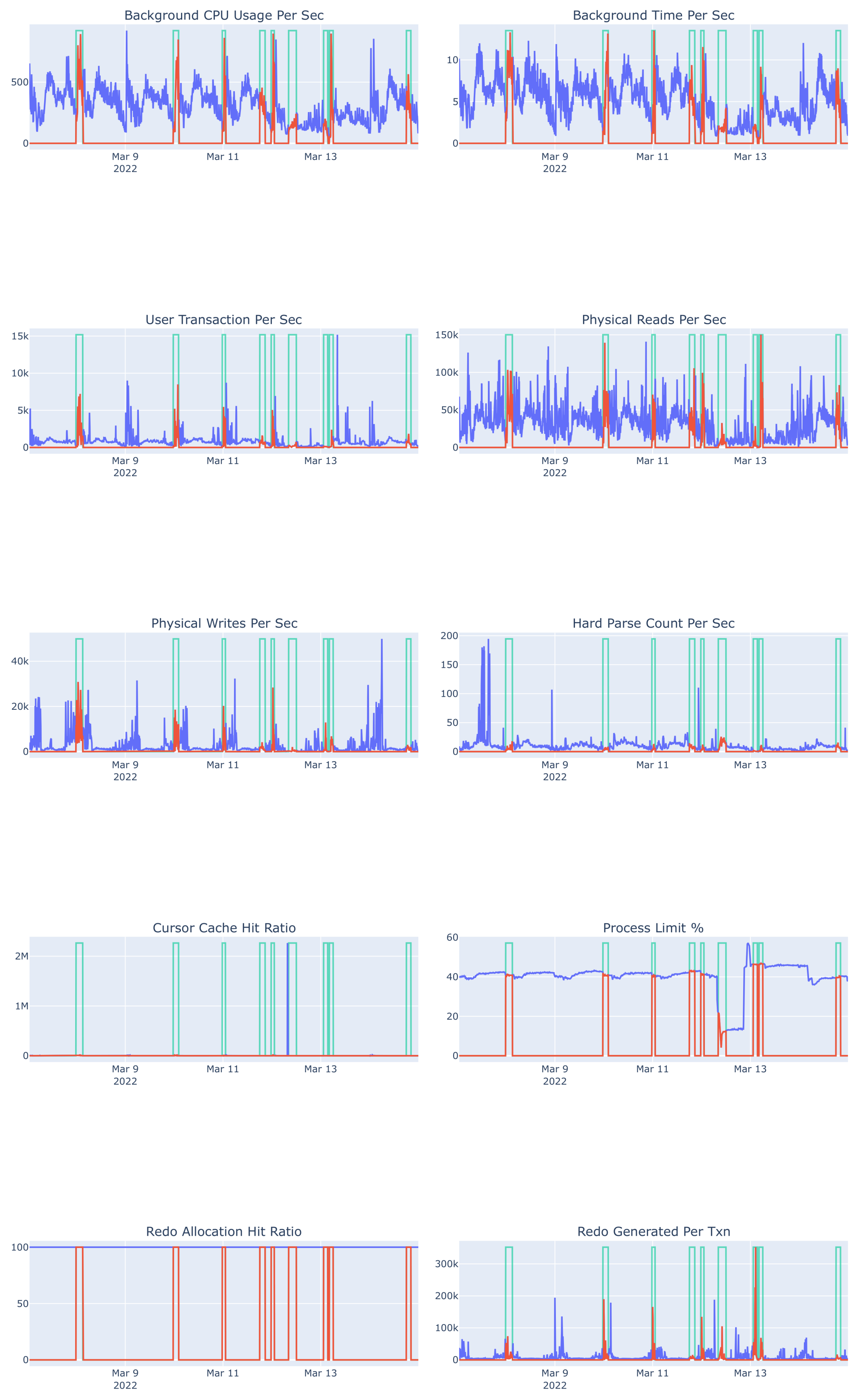


Figure 7: Metrics (Part 1) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.

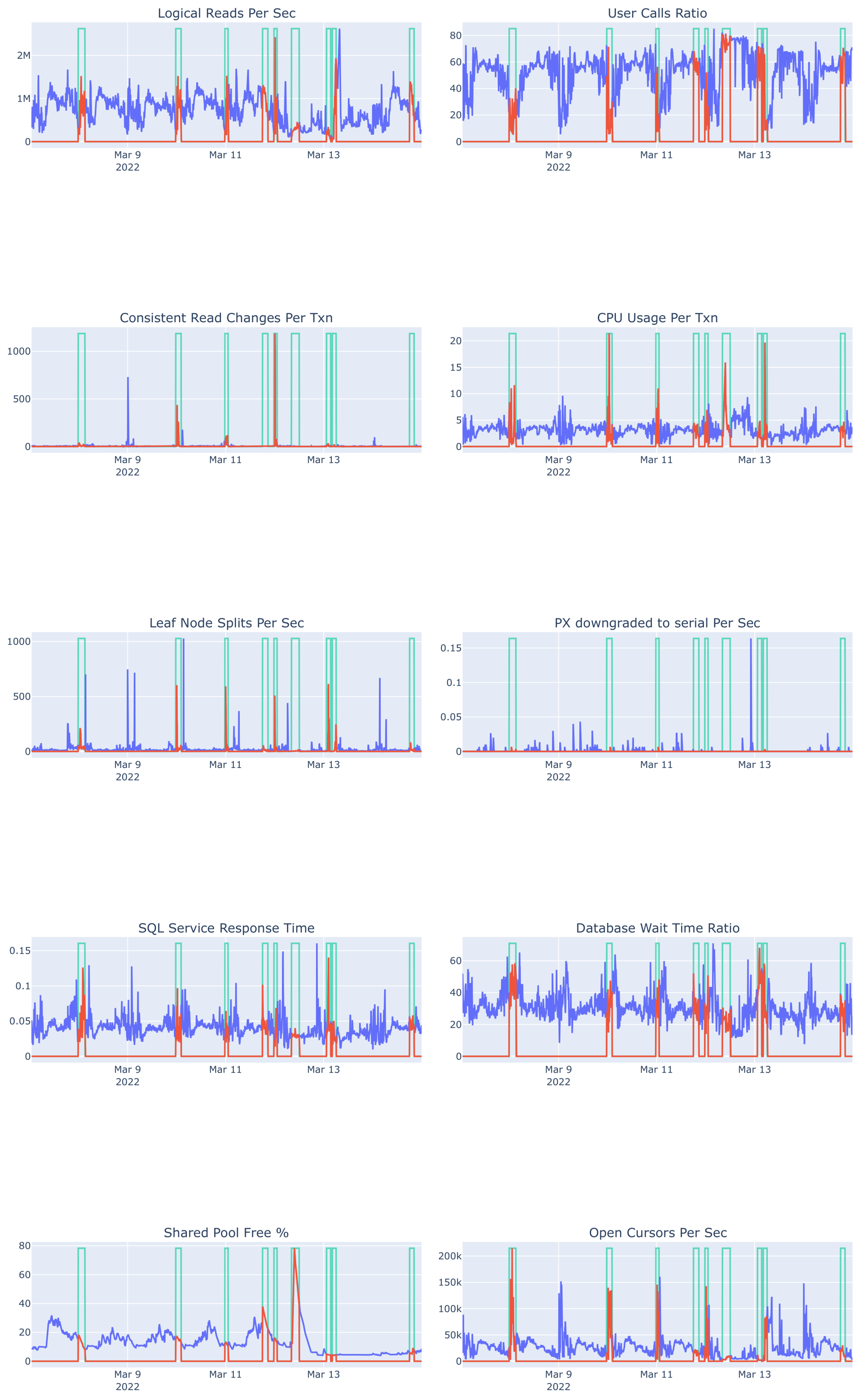


Figure 8: Metrics (Part 2) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.

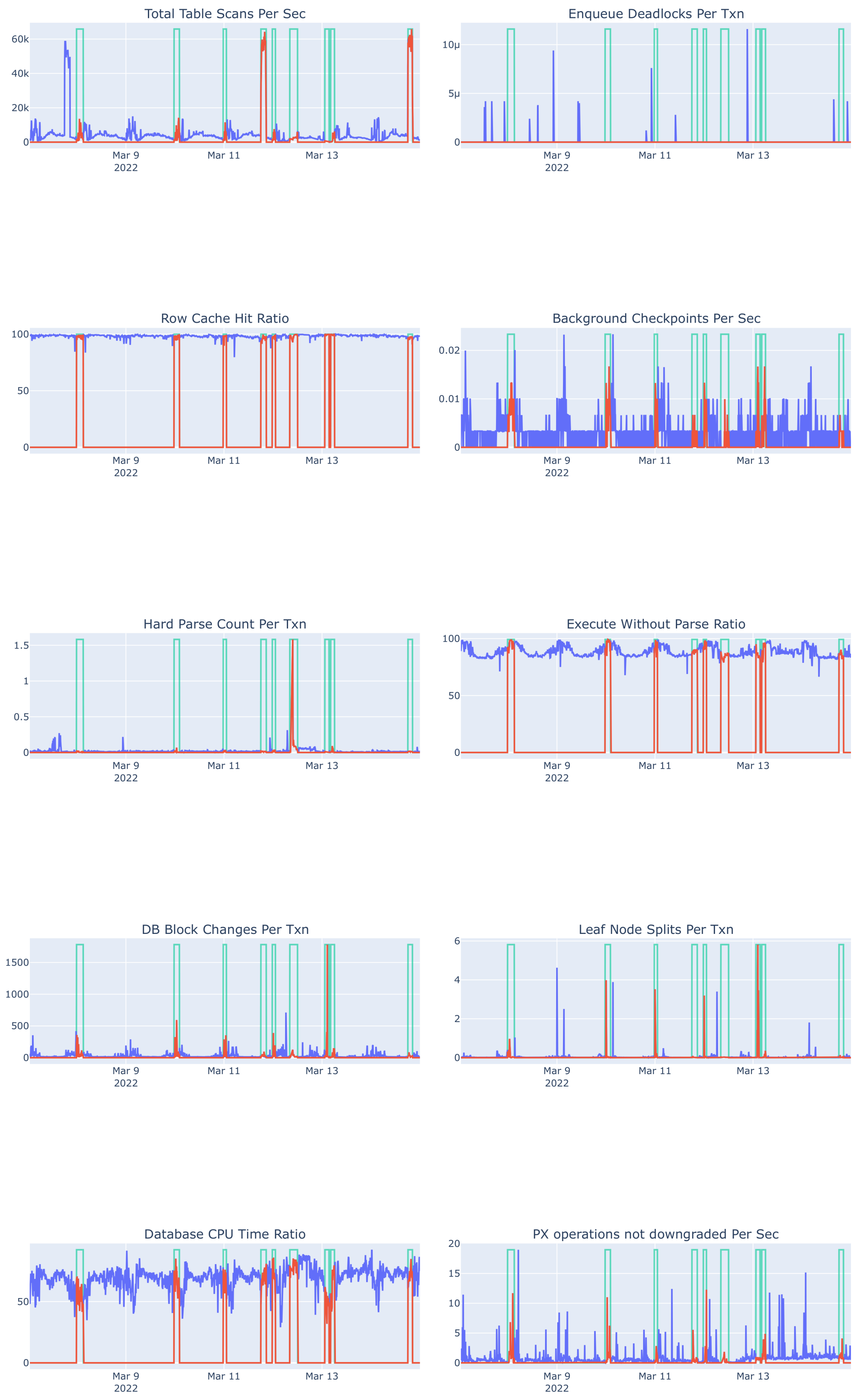


Figure 9: Metrics (Part 3) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.

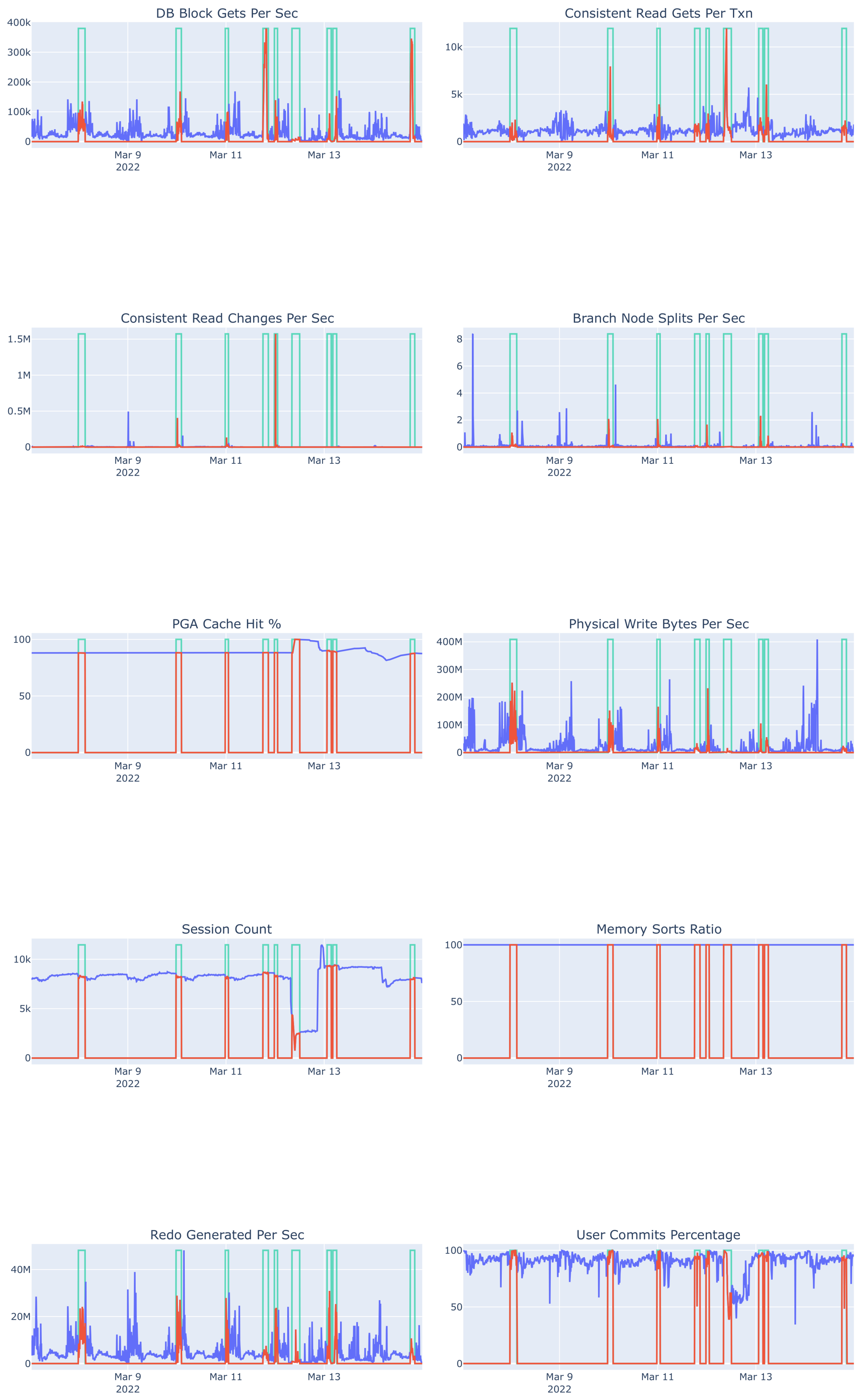


Figure 10: Metrics (Part 4) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.

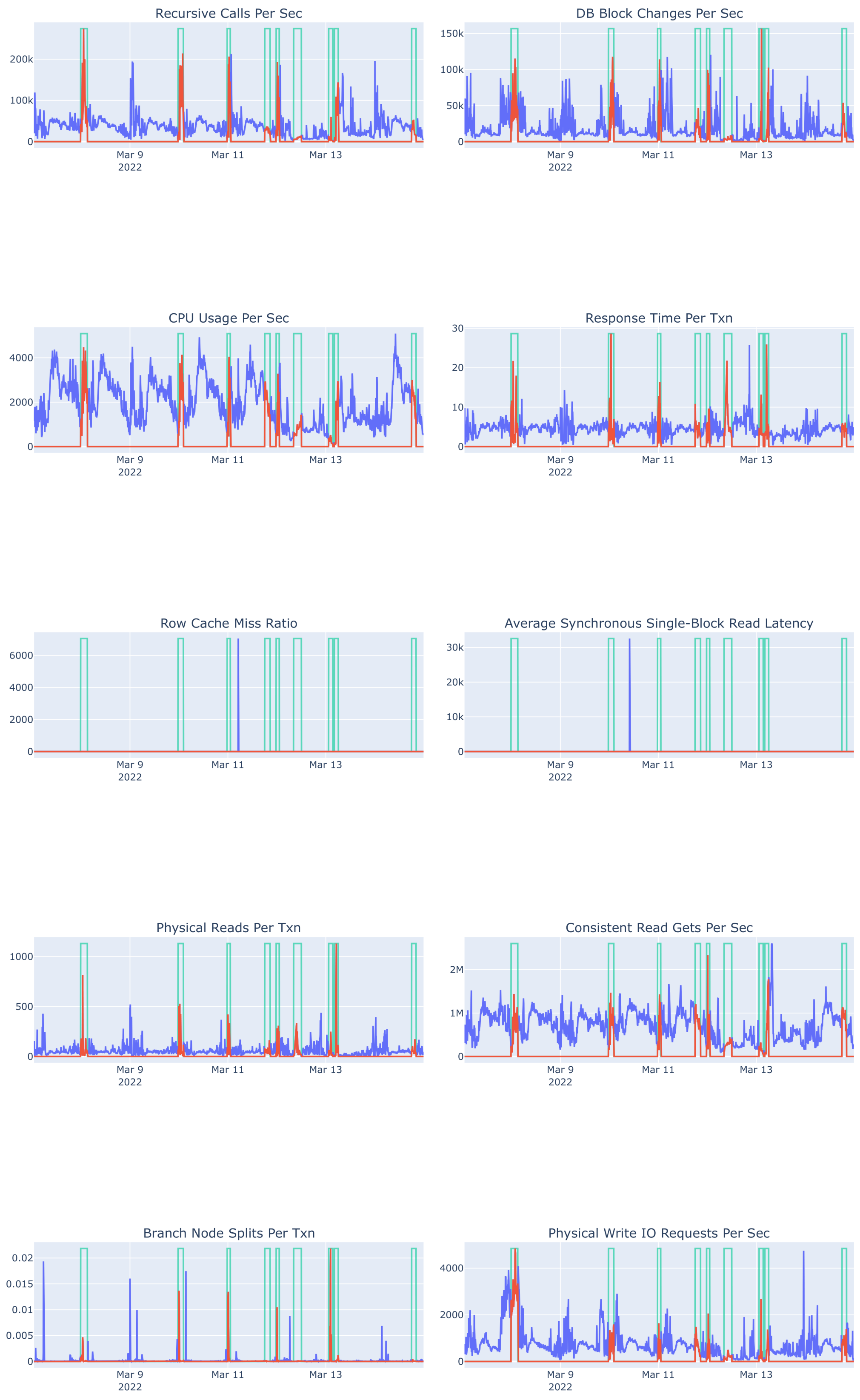


Figure 11: Metrics (Part 5) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.

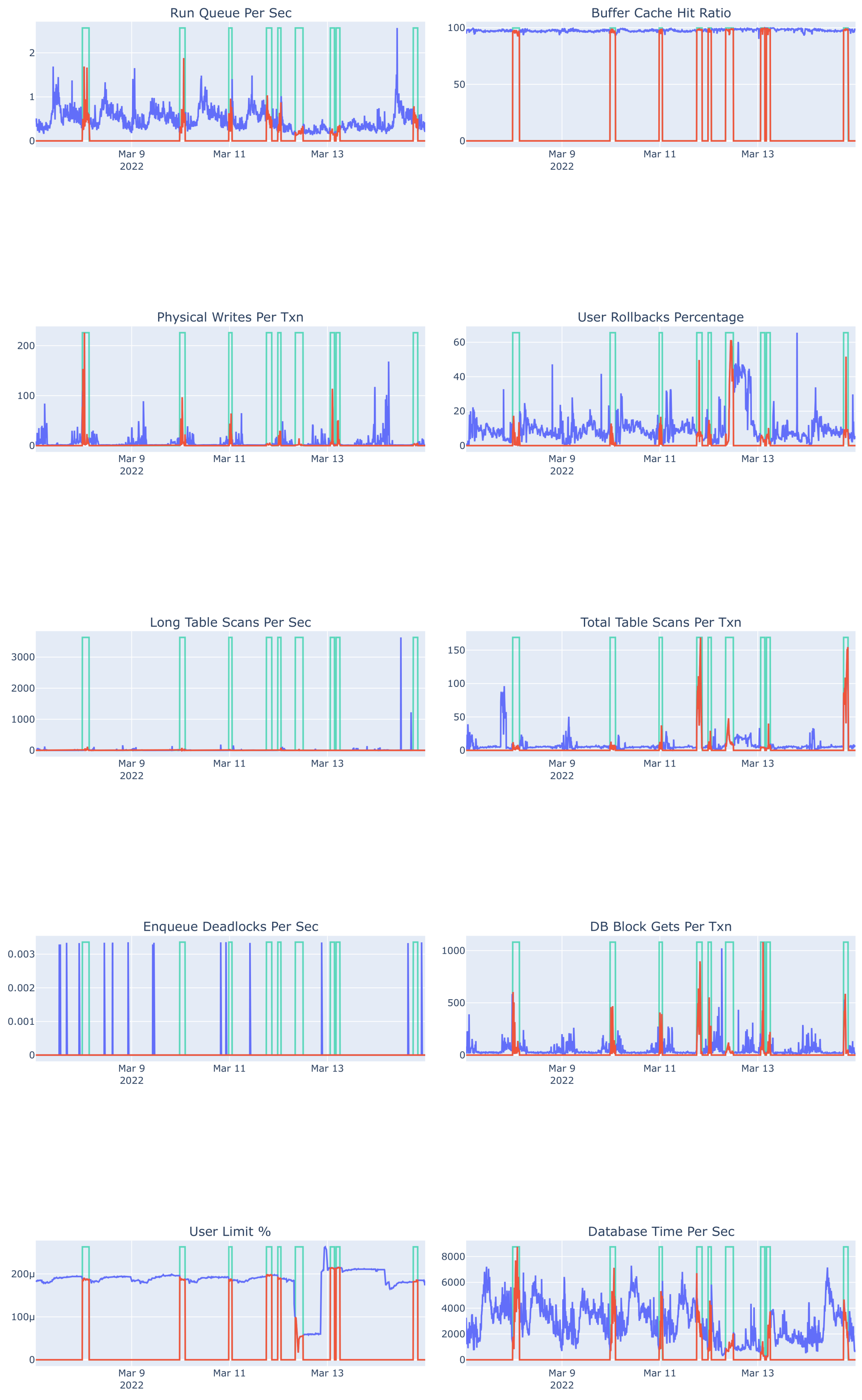


Figure 12: Metrics (Part 6) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.

B. Test data 2

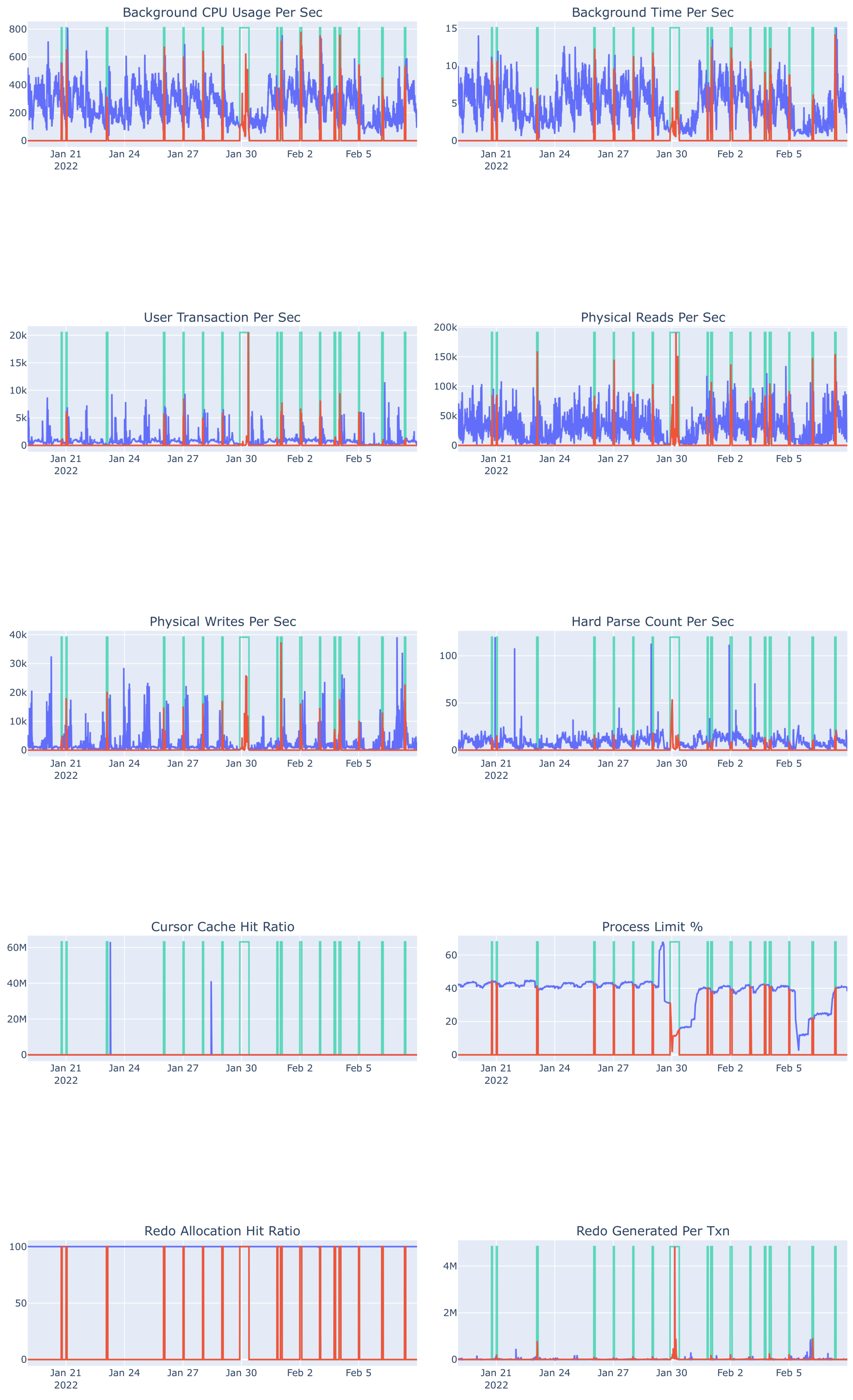


Figure 13: Metrics (Part 1) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.

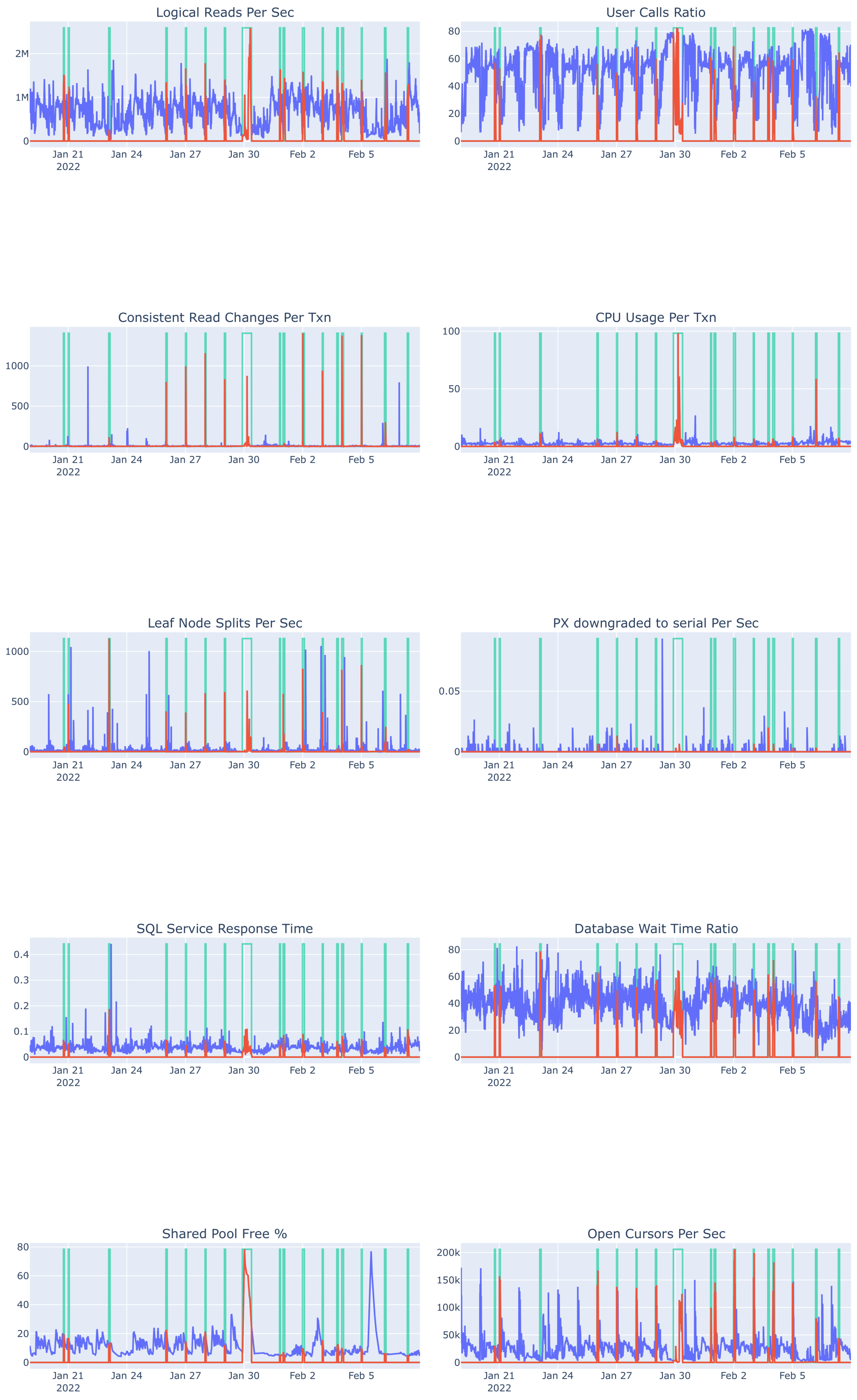


Figure 14: Metrics (Part 2) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.

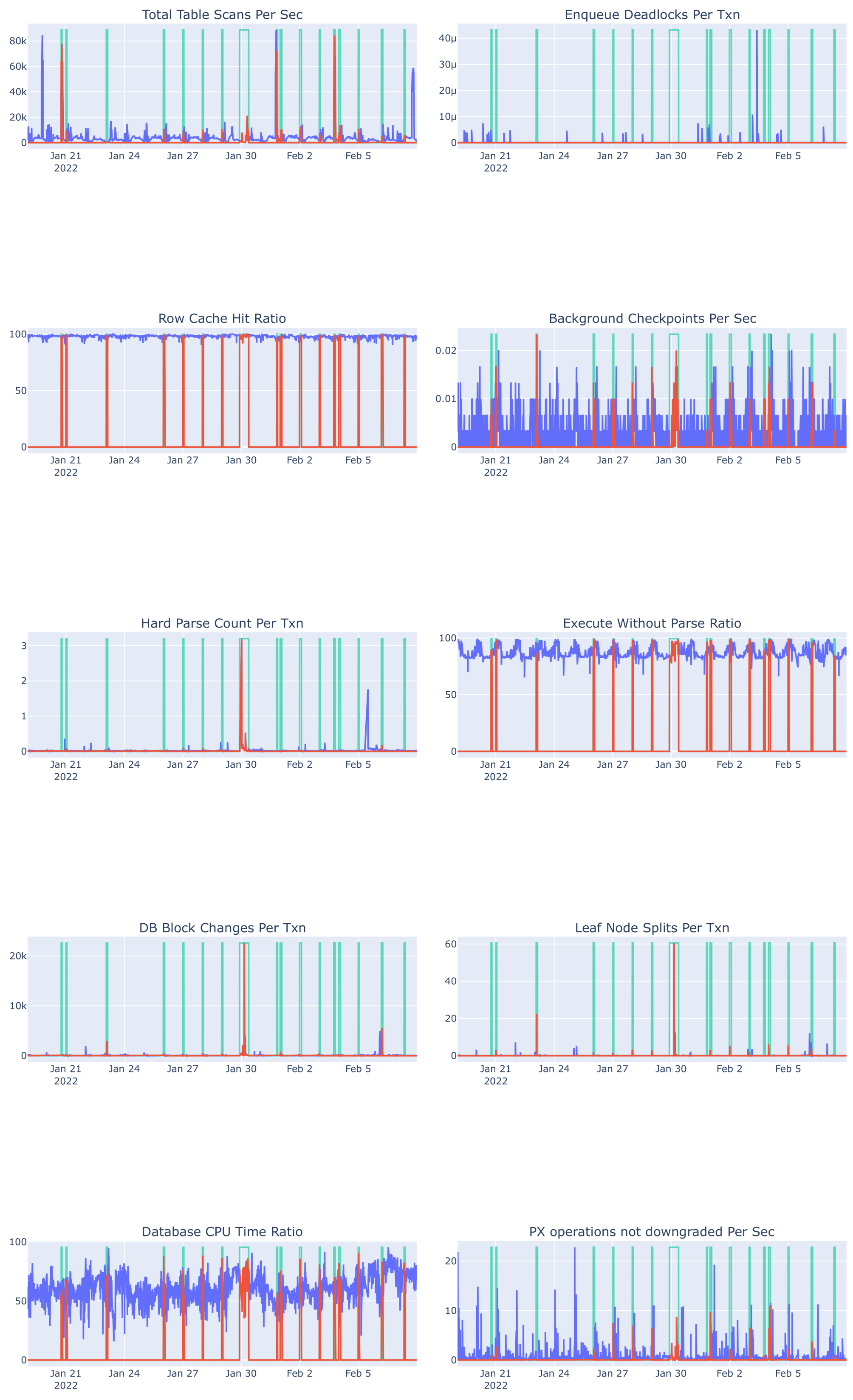


Figure 15: Metrics (Part 3) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.

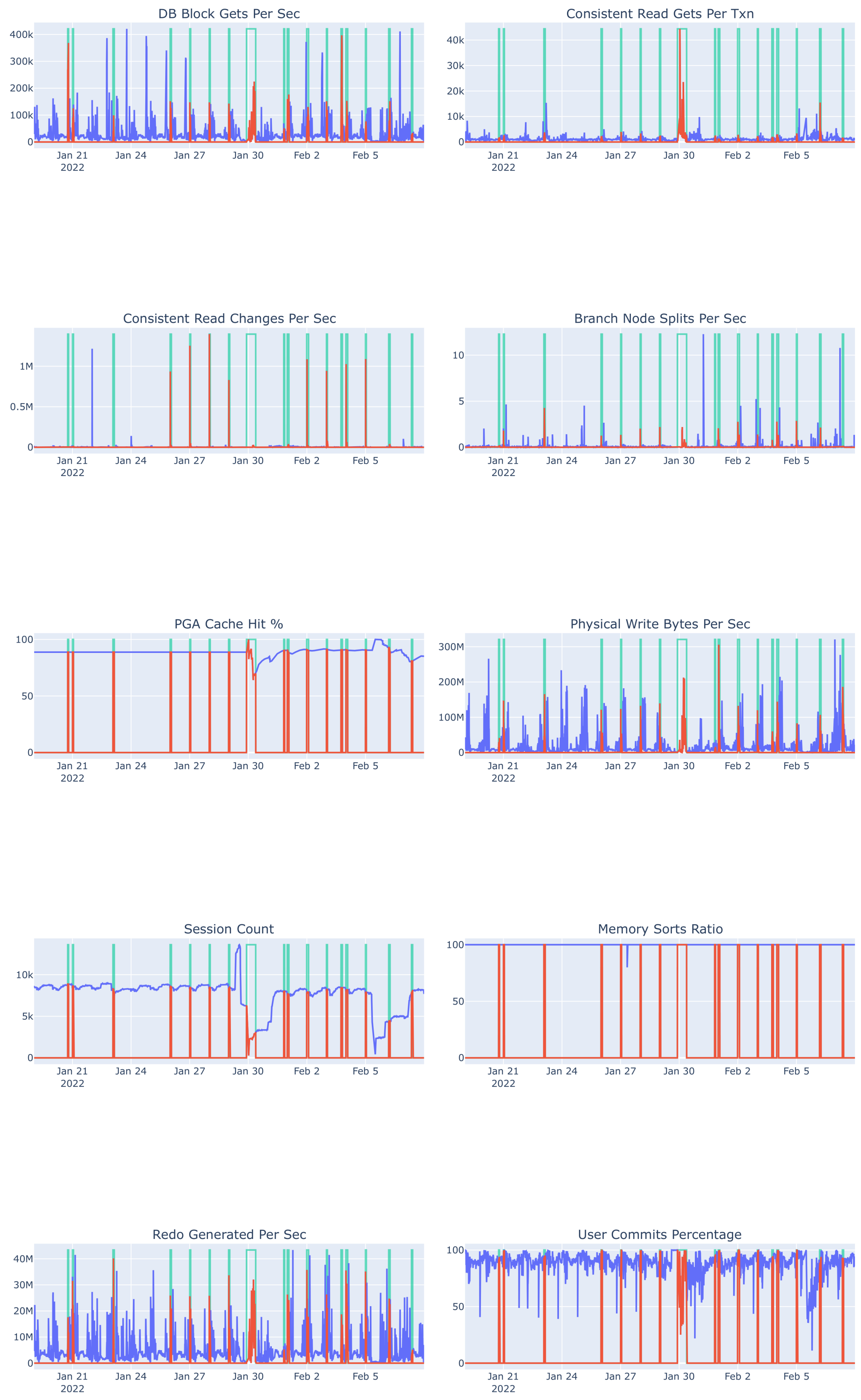


Figure 16: Metrics (Part 4) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.

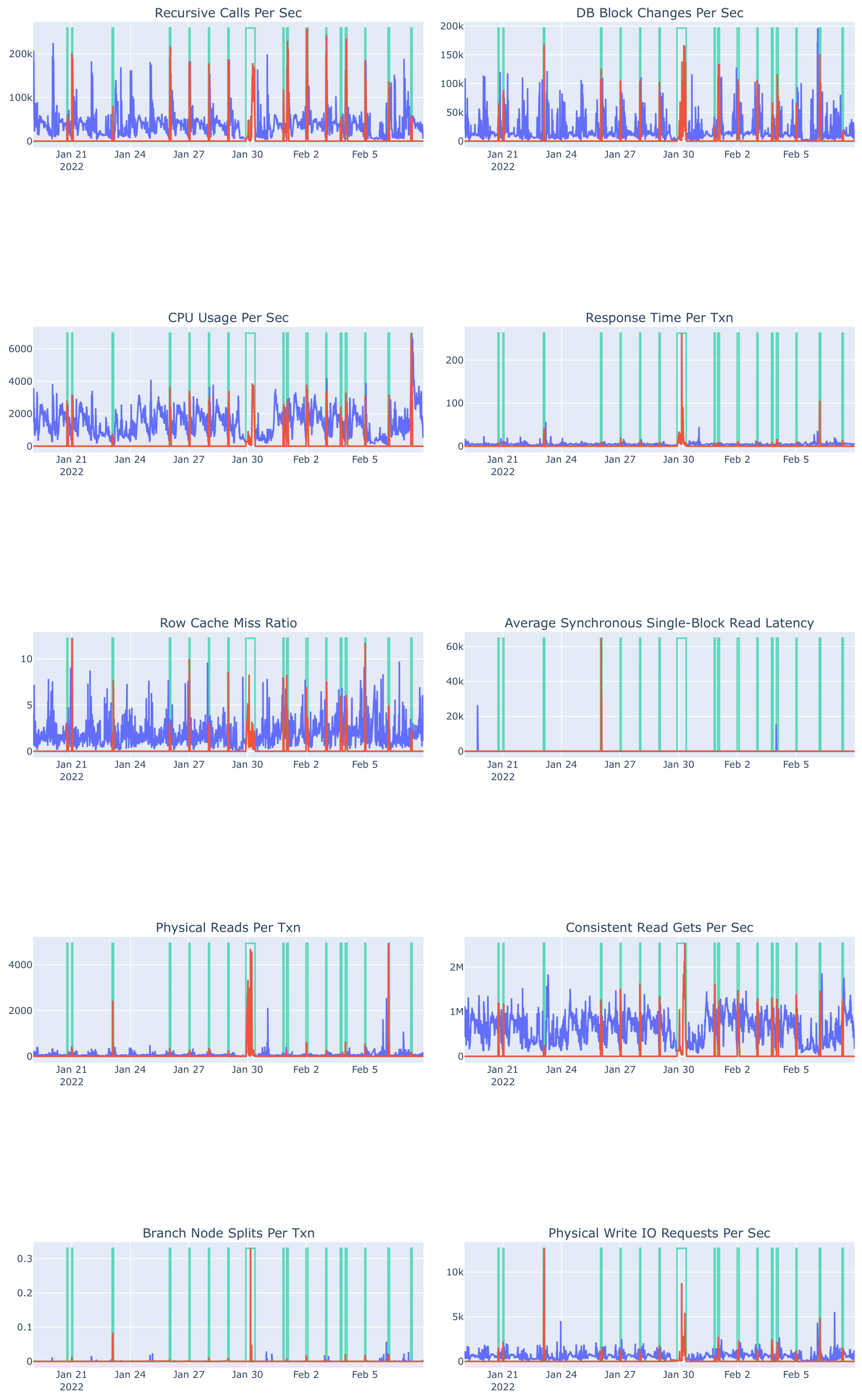


Figure 17: Metrics (Part 5) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.

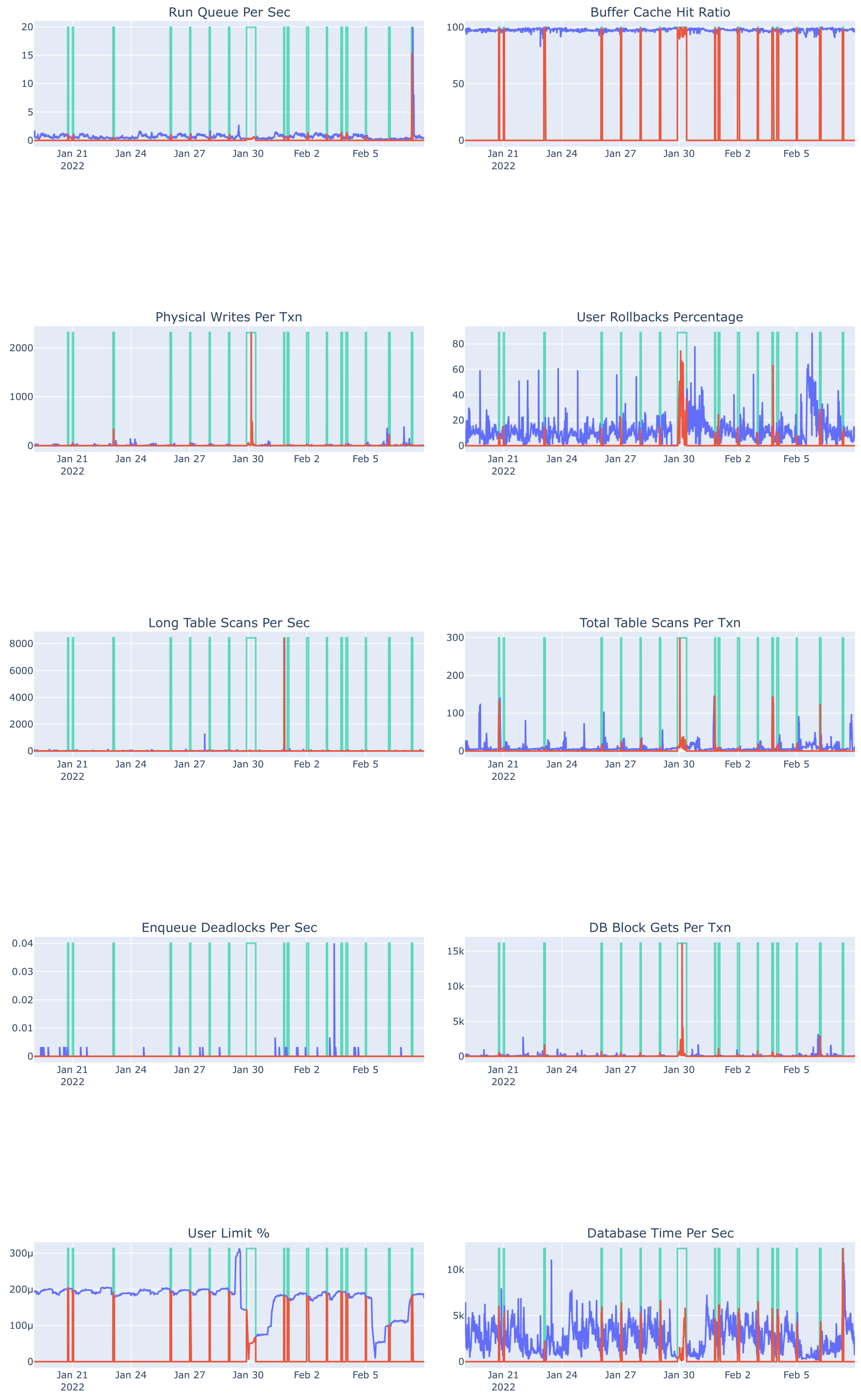


Figure 18: Metrics (Part 6) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.

Experiment 4:
Test on data from a different database instance

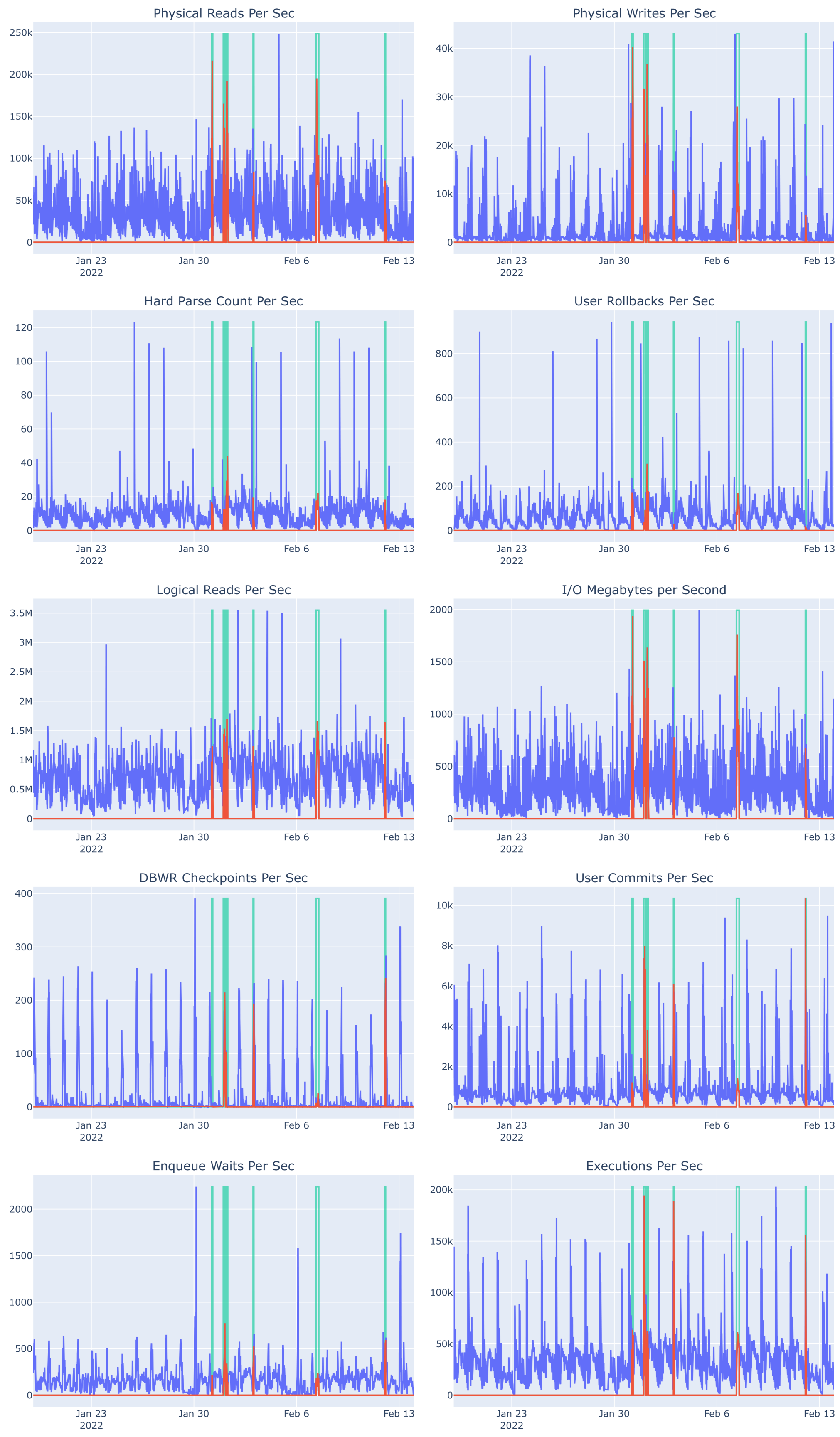


Figure 19: Metrics (Part 1) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.

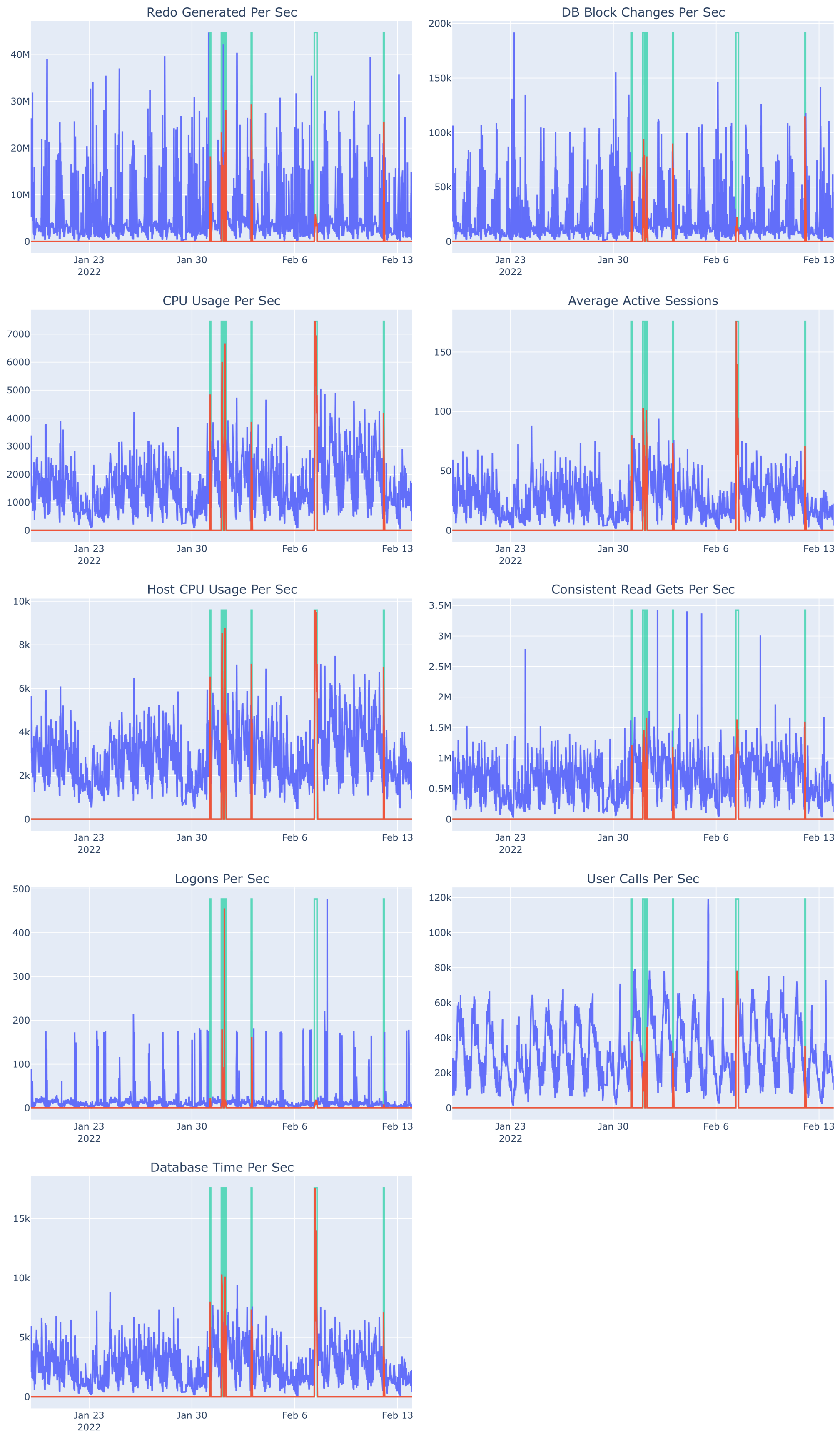


Figure 20: Metrics (Part 2) used for testing, the ports in green represents when an anomaly is detected. In red the value of each metric when an anomaly is detected.