NOVA IMS Information Management School

MGI

Mestrado em Gestão de Informação
Master Program in Information Management

Adding Kalman filter into ESI – An anomaly detection approach for middleware metadata

Leonardo Leal Corbisier

Internship Report presented as partial requirement for obtaining the master's degree in Information Management

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

Adding Kalman filter into ESI – An anomaly detection approach for middleware metadata

by

Leonardo Leal Corbisier

Internship Report presented as partial requirement for obtaining the master's degree in Information Management, with a specialization in Business Intelligence

Advisor / Co Advisor: Professor Doutor Mauro Castelli

January 2020

ABSTRACT

This report treats about the implementation of the Kalman filter algorithm into a product of Border Innovation - ESI – which had the goal to complement the range of algorithms available in the tool. Most specifically it treats about using the referred algorithm to point out anomalous events on a very particular Big-data scenario, which is the metadata about the data flowing in streaming through a middleware software. The focus of this report thus relies on providing reasoning about the problem which the algorithm has to deal with, and the data treatment adopted to implement the algorithm into the referred tool. It is relevant to mention that the practical implementation was done using Splunk, and therefore the technological aspects and the language of the tool were important factors that guided the algorithm implementation. The project took place during the author's work at the company and the data used to guide the project reflects the reality the tool is built to deal with, and it is fully anonymized to preserve the company interest.

KEYWORDS

Big-Data; Anomaly detection; Kalman filter; Border Innovation; Splunk.

INDEX

LIST OF ABBREVIATIONS AND ACRONYMS

Border Innovation – Border

Enterprise Service Bus – ESB

Enterprise Service Intelligence - ESI

Information technology – IT

Interquartile Range - IQR

Self-Organizing maps – SOM

Service-oriented Architecture - SOA

Support Vector Machine – SVM

Symbolic Aggregate Approximation – SAX

Business Process Automation – BPA

Simple object access protocol – SOAP

Representational state transfer - REST

JavaScript object notation - JSON

Extensible Markup Language – XLM

Splunk Process Language – SPL

Kalman Filter - KF

LIST OF FIGURES

1. INTRODUCTION

According to the Merriam-Webster dictionary (2018), an anomaly is something that differs from the common rule, that is abnormal, peculiar or not easily classified. Similarly, in statistics outliers are events far situated from the studied population. In real life, these phenomena are found as a hot day during winter or as an abnormally high number of clicks on a product web page.

From an IT infrastructure perspective, an anomaly may be a high volume of transaction requests or a slowdown on the transaction's runtime. While from a data-oriented point of view, an anomaly can be simplified as a data point or a series of data points that diverge from the common or expected behavior.

Detecting anomalies on an early stage provide the capacity to minimize the impact of a problem. For example, the development of a bottleneck on an IT infrastructure may cause systems unavailability for clients or webpages to be out of service. Furthermore, in a complex network of systems and interactions, it may take a long time to discover exactly where the problem is located, leading to undesired impacts. So, the earlier the anomaly is detected, the faster an action can be taken.

Different methods to detect anomalies can be used depending on the peculiarities of each individual circumstance. Important facts on choosing the best approach rely on factors like the ingestion and storage of the data, the level of noise it contains, the volume and the structure of the data and so on. Thus, in order to define the best model to be used, the properties of the problem and the data must be well known (Patcha & Park, 2007).

In cases where the data is static or historical common approaches vary from statistical to supervised or unsupervised learning. In these cases, algorithms like SOM, one-class SVM and others, are commonly applied. On the other hand, in cases of streaming data in which the sequence of the data points over time is a relevant factor, and the data pattern often evolves in the course of time, a good method to detect anomalies must be able to adapt to different parts of the stream in order to maintain a high detection accuracy (Tan & Ting, 2011). In such cases, some of the algorithms used are the RPCA, SAX, Twitter ADVec, Etsy Skyline and others (Ahmad, Lavin, Purdy, & Agha, 2017).

Considering the above and the practical business scenario where the present project fits, this work has the purpose to report the addition of the Kalman Filter algorithm into Border's existing anomaly detecting system for Big data, focusing specifically on the data retrieved from the IT infrastructure department of a Telecom company. For that, the report concentrates on the context where the system works, and the data manipulation needed to implement the algorithm into the tool.

The main reason why Kalman filter was chosen to be implemented into Border's system (ESI) is that the system is based in Splunk, and among the algorithms Splunk offers, Kalman filter showed to be the best option regarding the complexity of the scenario the system deals with.

Being so, it is also relevant to mention why among many possible approaches, Border decided to implement the one chosen. And the principal points are: firstly, the KF algorithm had to be adapted to an already working methodology of detecting anomalies; secondly because of the amount of data ingested into the system, which surpasses fifty new gigabytes of data per day; thirdly, the conception of the solution relied on the team expertise and know-how over the system and data; fourthly the

practical implementation had to be done in Splunk, and finally there was a deadline to implement the Kalman Filter into the system.

Splunk, in its instance, is a big-data analytical tool that allows the user to create queries, manipulate data and display data-analysis through different visualizations. It is also possible to build data-oriented apps on top of it. According to Splunk the software is capable of processing data from any format in any size and to dispose it in streaming to be analyzed. (Splunk, 2018).

1.1. BACKGROUND AND PROBLEM IDENTIFICATION

Border Innovation is an IT consulting company specialized in developing solutions around Service-Oriented architecture (SOA) and services Integration. One of the company's products is called Enterprise Service Intelligence (ESI) and it is a monitoring system for middleware metadata. It collects the metadata about the service flow within a company's middleware and performs several analyses over it, displaying them on a range of dashboards visualizations.

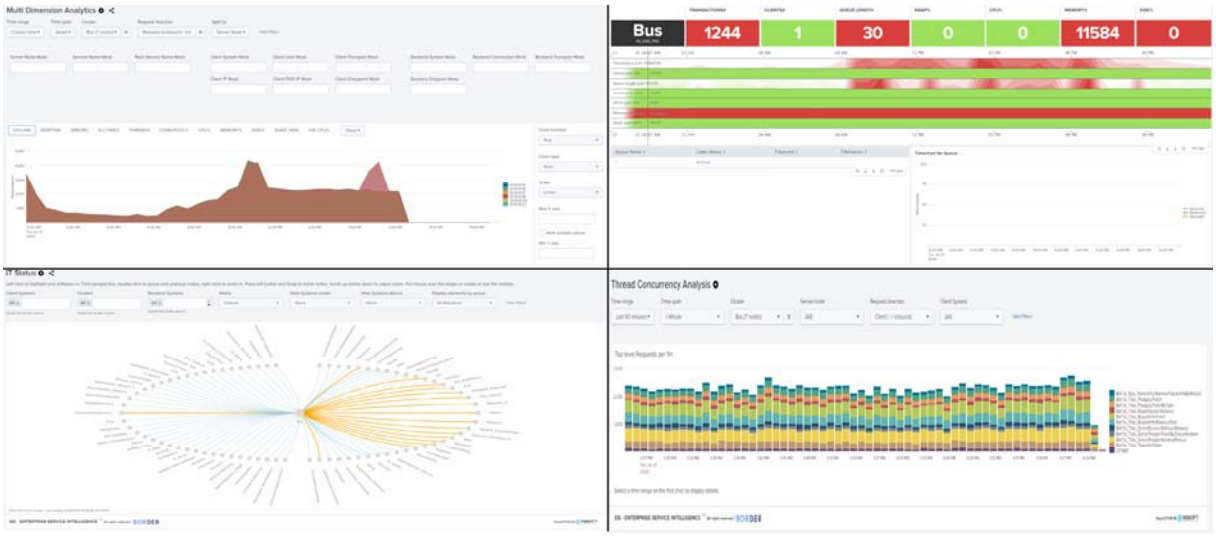

Figure 1 - ESI dashboards example

ESI's dashboards display information about the data flowing on a middleware, which in its instance composes a Big data scenario. once the data comes in streaming from several sources, in numerous formats and comprehends over fifty million new entry points per day.

Detecting anomalies is one type of analysis among all the ones the tool is able to perform. The present report thus covers just the anomaly detection component of ESI, which is just a brief extract of the whole system. Even more, the project focus on the work needed to insert a new algorithm to the system. Below there is an image of the anomaly analysis dashboard, which contains the visualizations generated by the respective ESI component.

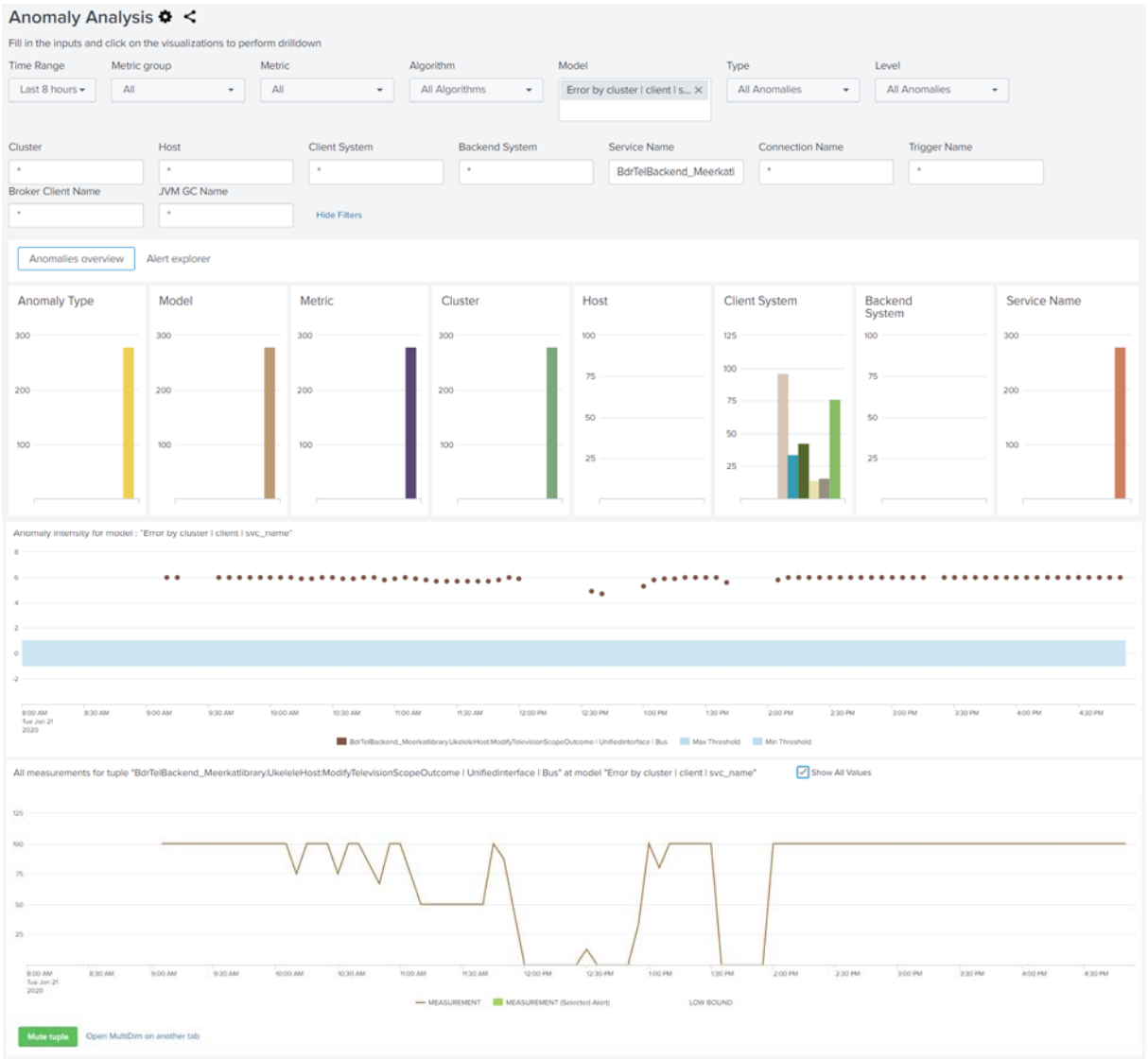

Figure 2 - Anomaly Analysis dashboard

The detection mechanism runs over measures which are called metrics, and when one or a series of metric points surpass the values considered acceptable by the system, these points are considered anomalous. This mechanism is periodically learning and creating new models on an unsupervised learning way, and therefore the implementation of this design requires the data to pass through different processing methods, which are explained in the present report.

In the context where ESI is inserted several metrics are analyzed, and each one of them requires specific attention in terms of data treatment. So, due to the limitations of the project scope just two different metrics are covered (volume and performance).

It is important to mention that for business protection purposes, the confidentiality of the data and strategical knowledge are not revealed, as neither as any other aspects that Border Innovation doesn't allow to be mentioned nor described.

Being so, the following objectives guide the accomplishment of the present project.

1.2. REPORT OBJECTIVES

The present work has the main goal to report the incorporation of the Kalman filter algorithm into the already existing ESI's anomaly detection component. More specifically, its emphasis relies on contextualizing the specific scenario where the system operates, and the steps adopted - in terms of data manipulation - to insert the algorithm as part of a system's detection model component. Therefore, the specific objectives of the project are:

1. Contextualize the scenario where the current report was held.
2. Report the steps adopted in terms of data processing to insert the Kalman filter algorithm into ESI

2. SCENARIO CONTEXTUALIZATION

To understand the data manipulation and adaptations needed to fit Kalman filter into ESI's anomaly detection mechanism it is fundamental to comprehend how a middleware works in terms of data transitioning, as well as the characteristics of ESI to which the algorithm needs to be adapted to. Also, it is important to provide contextualization about real case scenarios of the IT infrastructure department, so it becomes clear how the system operates and why an accurate detection system is valuable.

So, in order to proceed with the contextualization of the project report, this chapter brings in the first place concepts about middleware, followed by the data flow characteristics within such software, passing then through some metadata concepts and finalizing with practical examples of anomalies on a real case scenario, the It infrastructure department.

2.1.1. About ESI

ESI is a tool used to investigate the metadata of the data flow inside middleware software, providing thus visibility on the data transactions which happen inside the middleware. The tool is able to perform analyses over the data flow and to find bottlenecks between the systems' intercommunication. Most recently Border upgraded ESI by developing a component able to detect anomalies on the data flow. Unlike the rest of the components of ESI, which are built to be generical to any kind of data, the anomaly detection module requires some adaption to fit the particularities the data its deal with.

Being so, for the purpose of the present project the anomaly detection mechanism considers the scenario of the IT department of a Telecom company, and thus the inclusion of the Kalman filter algorithm is done based on the data observed in such environment.

So, before entering the data manipulation needed to apply Kalman filter algorithm for detecting anomalies, it is essential to provide some contextualization around middleware and how the data flows inside it. The contextualization is relevant because it allows the reader to understand the problem and to gain knowledge about the complexity of the scenario.

2.1.2. Middleware

A middleware is a software that stays between operational systems and the applications that run on them. It works as a hidden layer of connection and translation enabling the management and communication between distributed applications (Microsoft Azure, 2019). Mainly, it works by connecting different applications, by making the data from system available to others. A middleware serves to connect databases, application servers, web applications, transaction systems, monitoring systems and more.

“Middleware is the software that connects network-based requests generated by a client[1] to a back-end system. It is a general term for software that serves to "glue together" separate, often complex and already existing programs” (Margaret Rouse, 2019).

“With network-based interactions, a client, or requesting program, can make a request. That client is typically an application that resides on the front end, which is where the user interacts with software.” (Margaret Rouse, 2019). Examples of front-end clients are web browsers CRM and enterprise management systems.

“The back-end is the code that runs on the server, that receives requests from the clients, and contains the logic to send the appropriate data back to the client. The back-end also includes the database, which will persistently store all of the data for the application” (Back-End Web Architecture, 2020).

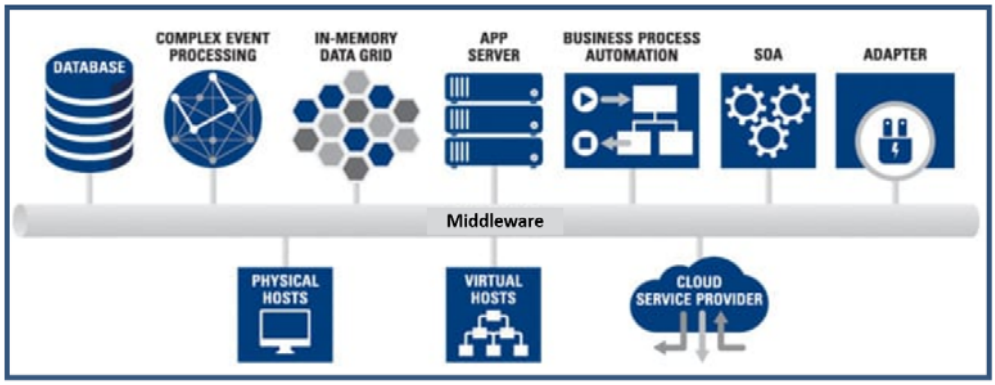

Figure 3 - Middleware representation[2]

The picture above is a representation of a middleware and the systems and applications it may connect, for example, databases, SAP, BPA systems, CRM’s, Siebel systems and so on. “The data stored on such programs can take many different forms and can be stored in many ways. Such as coming from a file server, fetched from a message queue or persisted in a database. The role of middleware is to enable and ease access to those back-end resources” (Margaret Rouse, 2019).

A middleware has different functionalities: besides serving as a connection hub between different back-end resources, it allows logic to be implemented over the data transaction, which expands thus its capability from just transferring data from point “A” to “B” to performing intelligent operations over data. For example, the middleware recognizes that the web browser requesting the data has its configuration set to English, and as a response to that it returns just English-based results.

Another important function of the middleware is to serve as a management tool for concurrent processing[3], load balancing[4] , and transaction management. The software is capable of distributing

[1] A client is a user or a requesting program that make requests to a server. For example, a web browser requests for pages from servers all over the web.

[2] Image adapted from https://sl.com/solutions/middleware-monitoring/

[3] “Concurrent computing is a form of computing in which several computations are executed during overlapping time periods - concurrently - instead of sequentially (one completing before the next starts)” (Rouse, 2019)

incoming transaction requests over multiple servers, virtual machines, and cloud zones by scaling its capacity vertically and horizontally. (Margaret Rouse, 2019).

Concerning the types of middleware, there are several different ones, and each one is used to fulfill specific functions with respect to the connection of applications, web and cloud services. According to Rouse (2019), among the most used kinds of middleware are:

- Messaging middleware: used for communication between distributed applications and services;
- Object middleware: enables software components or objects to interact and communicate with programs across distributed systems;
- Remote Procedure Call (RPC) middleware: used to provide protocols that allow a program to request a service from another program located on another computer or network;
- Database middleware: enables interactions and direct access to databases;
- Transactional middleware: uses transaction process monitoring to ensure that a transaction moves from one phase to the next one;
- Embedded middleware: facilitates communication and integration between real-time operating systems and embedded apps.

Despite its specificities, all the middleware categories have a common feature that is intercommunicating the distributed components connected to it. Therefore, being the channel that enables the data transitioning across different programs, the middleware creates a stream of data that flows from several points to various others. This happens with the data passing through the components inside the middleware and suffering the logical execution implied to them.

The dataflow is created between the different systems like so: first, a client-system makes a request to the middleware which than forwards the request to the back-end system. From the back-end system, the data is delivered to the middleware and forwarded further to the client-system that made the request. All the logical manipulation suffered by the data is done inside the middleware via operations called services.

[4] "Load balancing is a techniche used to distribute workloads uniformly across servers or other computer resources" (Alissa Irei, 2019)

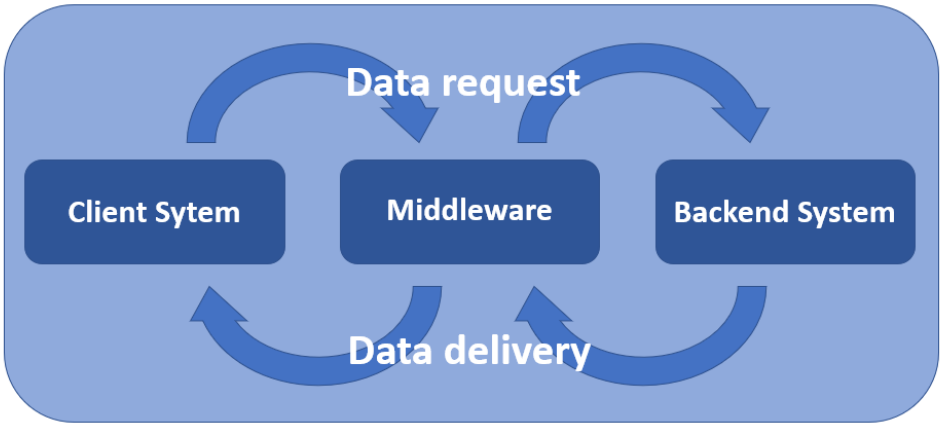

Figure 4 - Data request and delivery

2.1.3. Service execution on a middleware

The logic explained in the last paragraph - the requests, processing and delivery of data – occurs via message services, which are often based on web services, application services, and message tools. Such services support the management and the data transference in a distributed environment while providing interoperability between applications that run on different operating systems.

Regarding the different types of message services formats, the most commons are SOAP, REST, JSON and XLM. And the type of service used depends on the goal, restrictions of the service and the information communicated. A brief difference between the message services formats, according to Raygun (2019), is listed below:

- SOAP is a standardized protocol that has strict rules, advanced security features, high complexity and requires larger bandwidth and resources when compared to REST. It is often used when the job executed by the service requires high security and complex transactions, such as for financial transactions, CRM software, payment gateways, and others.

- REST is not a protocol but an architectural style, and it was created to address the problems of SOAP. Therefore, it is more flexible, lightweight and has better performance than SOAP. It also allows different message formats like HTML, JSON, XML and plain text, while SOAP only allows XML. REST can be used for example to navigate to a web page from an index, or in cases where execution performance is a priority.

- JSON is an easy-to-parse and lightweight data-interchange format. Its format structure consists of collections of name/value pairs and ordered lists of values. These are universal data structures used by most programming languages, which allows JSON to be easily integrated by different languages.

- XML is a flexible way to share structured data via network. Like JSON it describes the content of the data it contains.

Message services serve to execute a job and are basically user-configured operations and transactions over data. Through a service, the user chooses where the data should be picked from, the manipulations it should suffer and to which systems it should be delivered to.

A simple example of a service invocation is shown below. It illustrates how the data flows on a similar, but simpler, scenario to the one where the project was executed. To maintain the level of complexity as simple as possible, the components represented on the image are just the three main ones: a client-system, a middleware, and a back-end system.

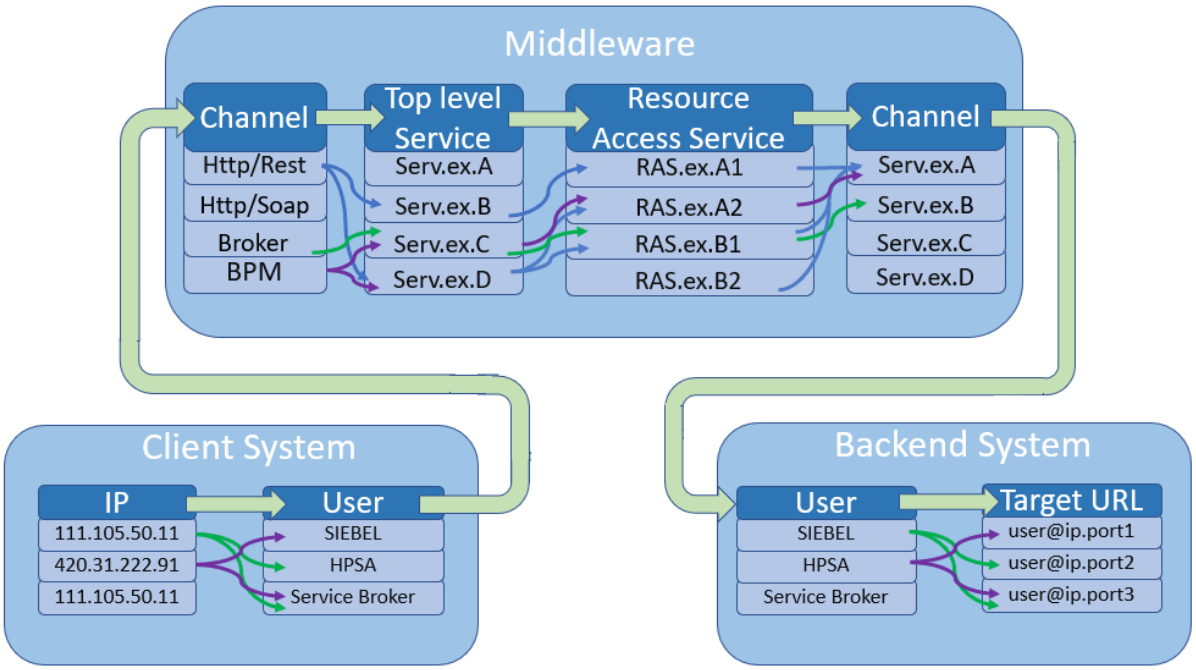

Figure 5 – Data request flow

The service invocation is triggered by a client system, which in its instance is identified by an IP address and a provisioning tool (also called user), to which the request is actioned to. The provisioning tool sets up the service to be processed by the middleware. The request is then forwarded to the middleware and there the channel is chosen according to the type of service the request is related to. Then, one or multiple services are executed, hitting finally the backend system. Once the service hits the backend system, the accordant system which contains the data and the path to the data is actioned. This briefly resumes the data request journey.

The data delivery journey is performed in the opposite direction. Starting from the back-end system and passing through the components actioned for the specific data transaction. The final situation of the transaction is to provide the client system with the data it requested. So basically, a client-system "A" asks for data stored on a back-end system "B" by triggering a service request to the middleware, which in its instance forwards the request to a back-end system. From there, the target back-end system retrieves the data and forwards it to the service inside the middleware which forwards the result to the client system.

The image below shows the data delivery journey.

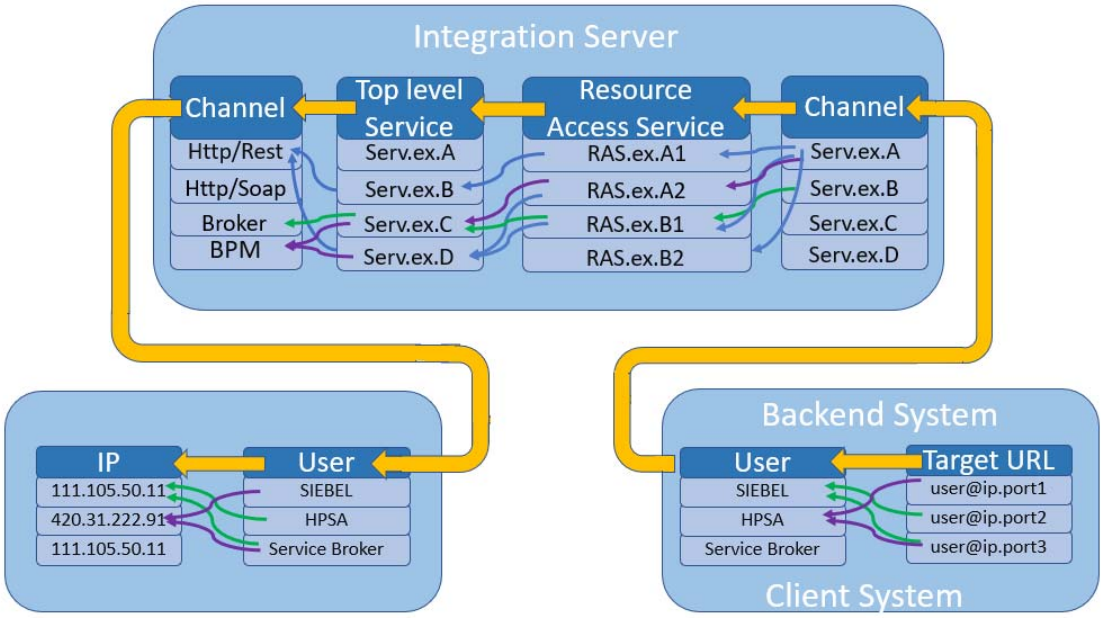

Figure 6 - Data delivery flow

The complexity of this logic increases when the infrastructure where the middleware is inserted takes shape of a network, and thus several client and back-end systems, and many middlewares take part in it. When mediating data transactions on environments with several systems, different message services compete for process capacity while being evoked and triggered by numerous clients and back-end systems simultaneously.

Also, in a complex environment, a single back-end system is requested from multiple services at the same time, and thus provides data to multiple client requests in a tiny fraction of time. The high volume of transactions per time creates the need for other components like load balancers and queue management systems. These components though are out of the scope in this project and therefore they will not be addressed.

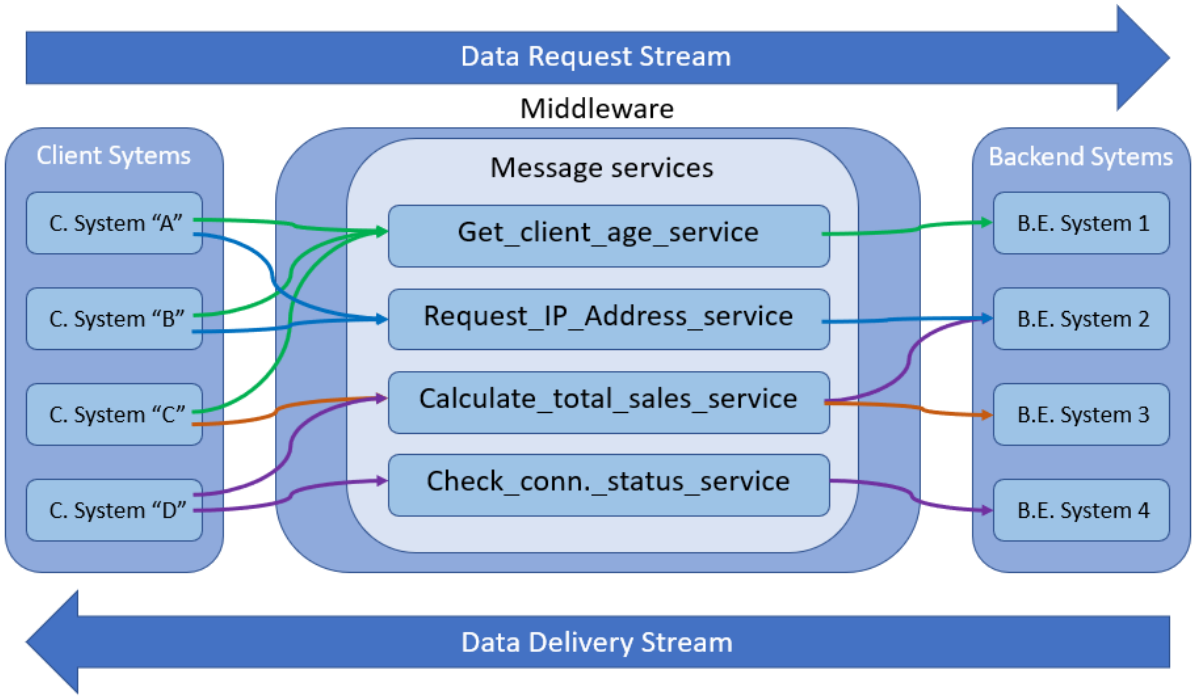

Figure 7 - Data request stream on a middleware

The Image above illustrates examples where multiple service requests are running concomitantly. It shows one (or multiple) client systems triggering the same (or different) message services, and the message services requesting data from different (or a single) back-end systems. On the Image, the data request stream flows from left to right and the data delivery stream flows from the right to the left. The small colored arrows represent the data requests.

The data supply related to the requests of the picture above is represented on the image below, so on the following picture, the data delivery is the stream in focus. It shows the multiple back-end systems supplying services which in its stance return the data to the client systems. Again, multiple combinations of back-end systems/services/client systems are possible.

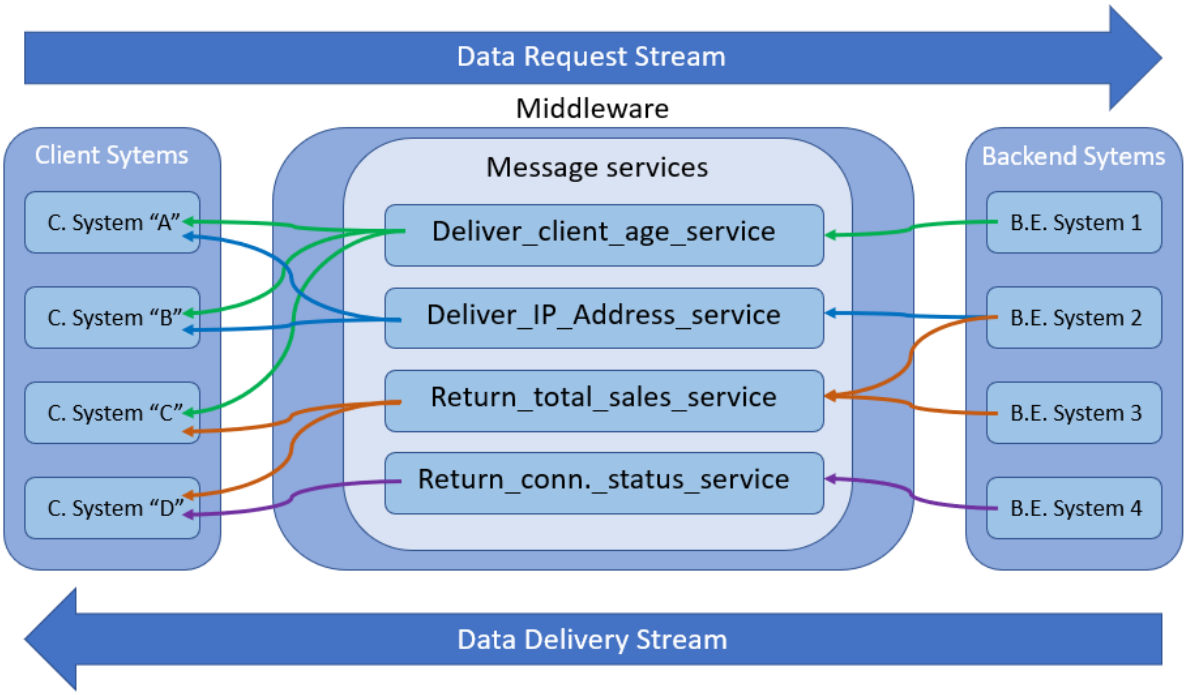

Figure 8 Data delivery stream on a middleware

On the infrastructure where the project was held, there are several other nature of services: services that take data from one back-end system and insert it on another back-end system; services where the client system just writes the data on the back-end system; and many other types of combinations regarding the data request/delivery. It is also common to have thousands of services running simultaneously and automatically, nested services, asynchronous services and many other variations that increase exponentially the complexity of the data transaction.

The described process of data request and delivery produces traces. By looking into the services metadata, it is possible to see its name, which back-end and client systems were involved in the transaction, when it was executed, how long the service took to be executed[5], and more. From that, innumerous metrics can be calculated. For example, how many different services were executed at the same time or how often a service was executed in a given period[6].

Since the tracked data path is done by analyzing the middleware metadata, the next section brings a brief explanation of a few metadata concepts.

[5] Named Performance
[6] Named Volume

2.1.4. Metadata

Metadata is commonly described as “data about data”. Or according to Caplan (2003) metadata is data that provides information or describes other data. Riley (2017) goes further affirming that metadata is the information created, stored and shared to describe things, allowing people to interact with these things and to obtain the knowledge about it.

Different authors classify metadata into different categories. Most commonly found definitions are, for instance:

- Administrative Metadata: data related to file management, rights and intellectual property, permissions, etc.;
- Descriptive metadata: data used to find, describe and understand an element. For instance, a title;
- Statistical metadata: data which produces statistical data;
- Structural Metadata: data about the relationship of one resource to one another.

In the case of ESI, all the metadata categories mentioned above take part in the analysis. The system uses all those metadata types to identify relationships on the data, calculate statistics and describe what is acceptable in terms of measurements.

The statistical metadata is particularly relevant for the ESI once this is the data measured to classify one event as an anomaly or not. In this report, for nomenclature purposes, the statistical metadata is called metrics. So, each metric represents a different type of statistical metadata – or calculation over such metadata type. Which, in its instance, describes one aspect of the middleware state, more specifically about the service invocation state. The metrics target of this report are performance and volume.

The performance metric informs how long a service request took to be executed, and the volume means how many invocations of a service were done on a period of time.

Not less important is the descriptive metadata, once the abnormal value of a metric is identified to happen on a certain combination of components. The components – back-end system, client system, and service name, are also named dimensions, and the unique combinations of these dimensions are called tuple.

Thus, an anomaly is always associated to a metric and a tuple. Being the first, the aspect on which the numerical value is measured, and the second the name that identifies the components involved in the measurement where the anomaly is present. In the next chapter, the concepts around metrics and tuples will become easier to understand once practical examples will be given.

Now that the theoretical concepts about middleware and the data analyzed by ESI were clarified, the next section focuses on providing a clearer understanding of the practical scenario by bringing real case examples.

2.1.5. Practical examples

Before explaining the mechanism developed to detect outliers on the previously described scenario, this chapter brings real examples of the anomalous events on ESI, with the intention to provide more knowledge depth and better visualization about the project as a whole.

It is already known that through the services metadata the internal state of the middleware is informed, and from there it's possible to generate a range of metrics that can be analyzed to measure and monitor important situations. So, in short, the metrics reflect aspects of the data flow characteristic inside the middleware and inform a tracked path from where a data point passed through.

Since all the components involved in the data's path are suitable to face technological difficulties, all of them may be possible roots of an anomaly. By keeping track of the metrics and knowing its data trail, it's possible to get to each component involved in the transaction isolated and discover exactly where the problem is located.

As mentioned before, the components involved in the data flow are named dimensions, and the unique combinations of dimensions are called tuples. Over the dimensions, slices and drill-downs can be performed, and because of the limited scope of the project just the service name, the back-end system, and client system dimensions are approached in the project.

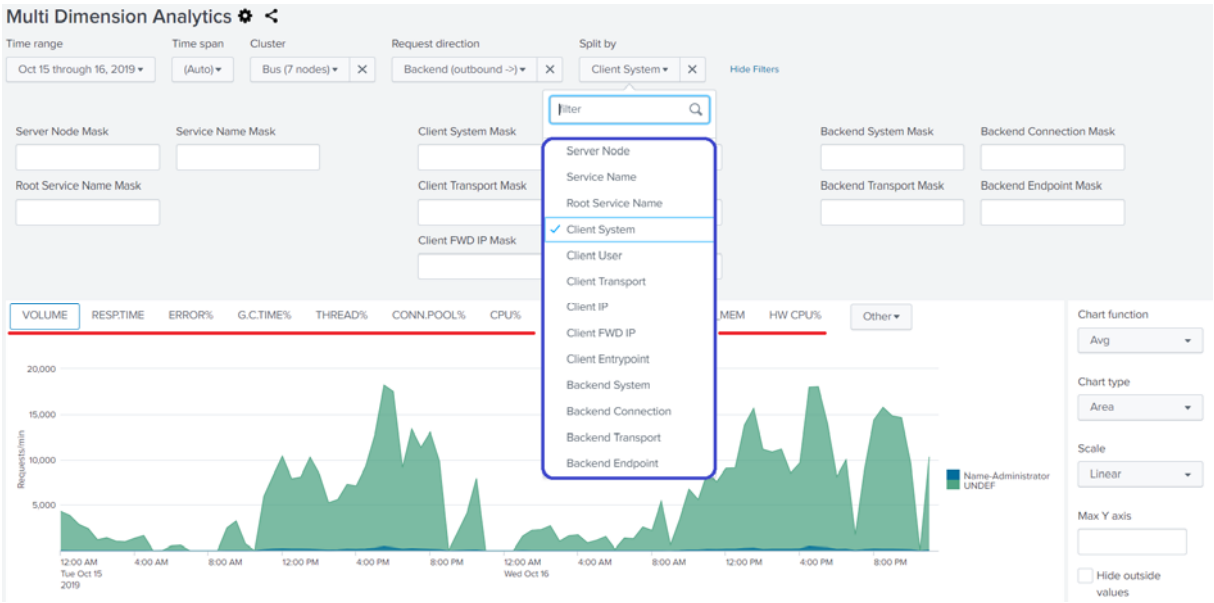

Figure 9 – Dimensions on ESI's Multi dimension analytics Dashboard

The previous image is a screenshot of the ESI Multi dimension analytics dashboard, and it shows the dimensions circled in blue and the metrics underlined in red. The graphic represents the volume of services invoked by all the client systems - two in this case: Name-Administrator and UNDEF - on a determined period.

By choosing another dimension, such as the service the name, the volume by each service name is shown on the graphic, like on the next image.

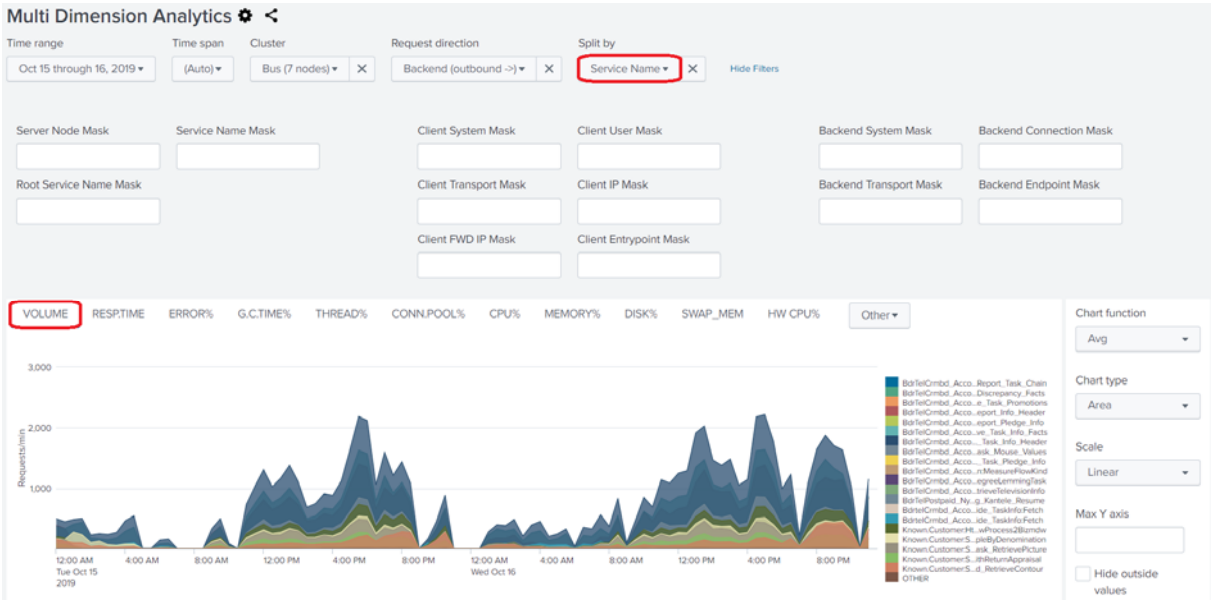

Figure 10 - Volume by service name

The metric volume represents the total number of service requests per minute. And on the graphic above the different colors represent different services.

From there by choosing one specific service and another dimension like the back-end system, a drill-down shows the volume by each back-end system that supplied that specific service. On the next image is visible that two back-end systems (Authenticator and UNDEF) supplied the service "Know.Customer.Soap…" during October 15th and 16th.

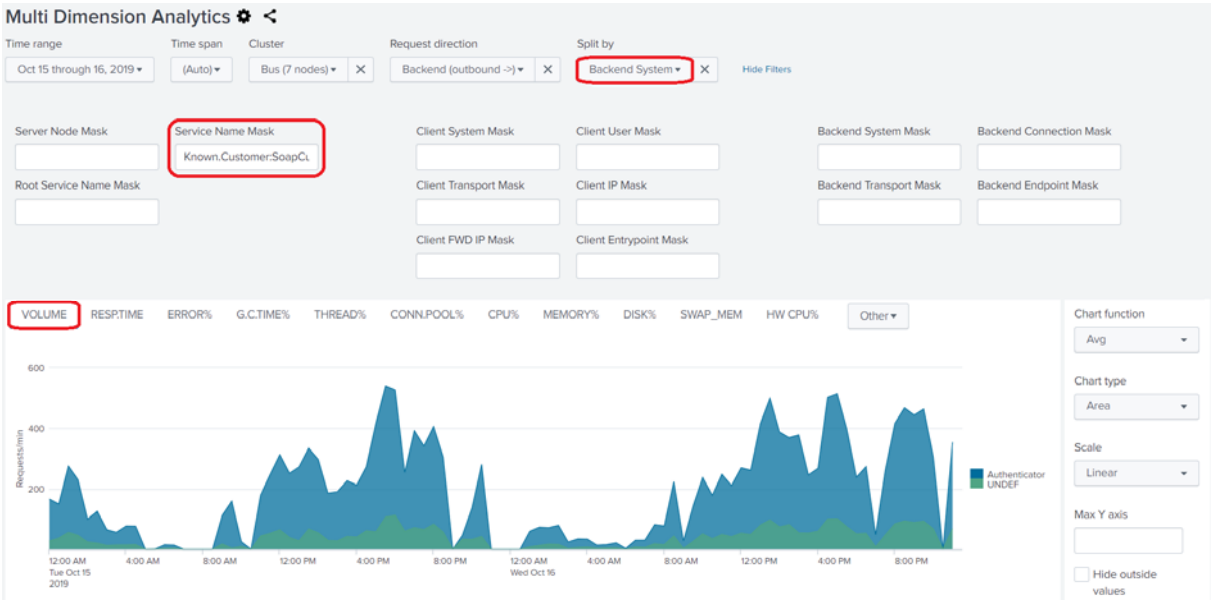

Figure 11 - Volume by service and back-end

In such a scenario there are innumerous combinations of analysis that can be performed. Above, analysis of volume was shown, and bellow graphics of performance are visible.

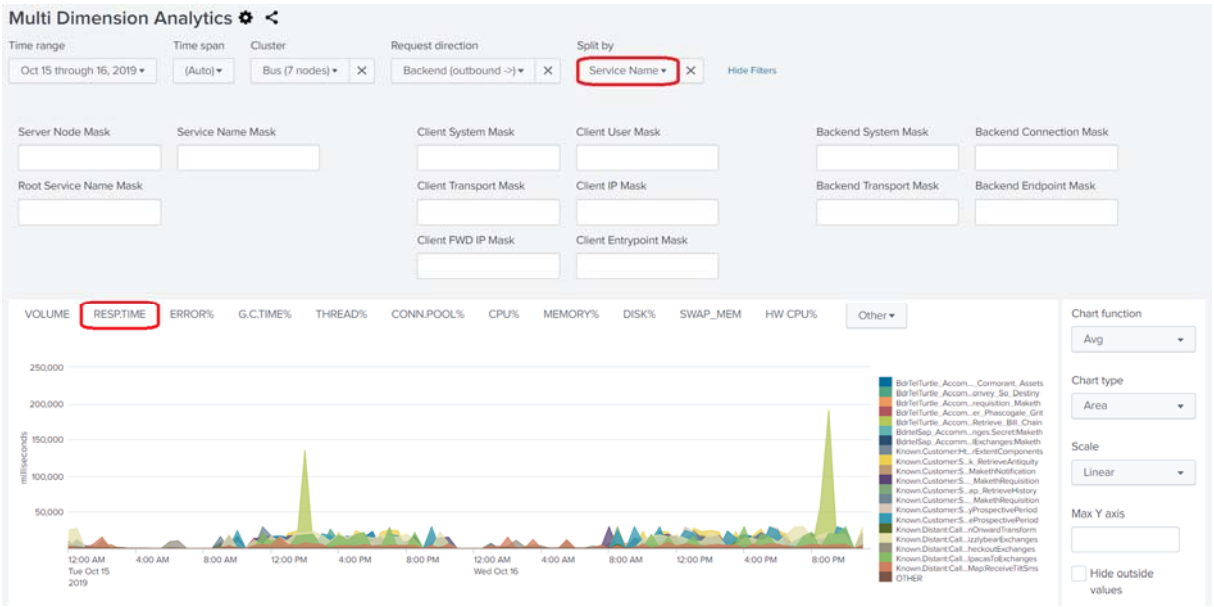

Figure 12 - Figure 10 - Performance by service name

the performance by service during October 15th and October 16th is shown above. Again, all the services invocated in this period are shown in different colors. The corresponding drill-down by a specific service and by back-end system for performance is shown next:

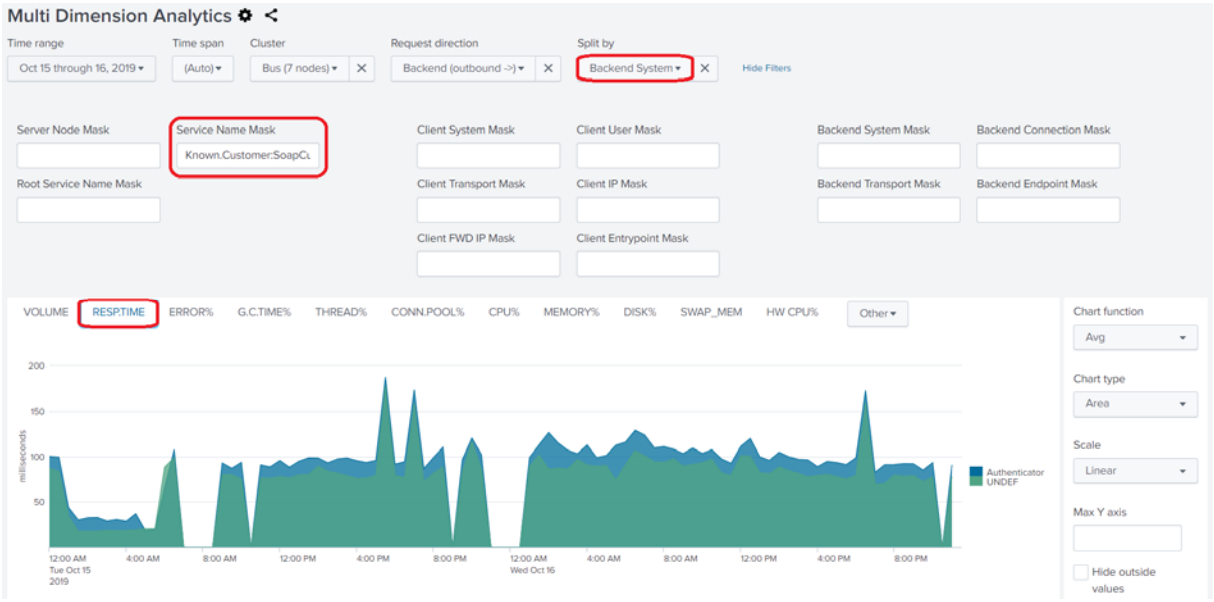

Figure 13 - Performance by back-end system

The graphic above shows all the back-end systems (Authenticator and UNDEF) involved in the transactions requested by the service “Known.Customer.Soap…”

In the context of the IT infrastructure, these visualizations are relevant once they show the status of the components involved in the data transaction and enable comparisons between them. Even more, when a breakdown or a bottleneck situation happens it becomes visually clear and it allows the identification of the components involved in the event.

Bellow, the first image illustrates a breakdown on all the servers in terms of volume, which means that all the services requests drastically decreased on all the servers. Further analysis over the other metrics revealed that a couple of services clog the servers which caused the connection pools usage and of the thread components to drastically increase in all the servers.

The purpose of the image is just to illustrate a real breakdown situation and therefore the meaning of the metrics is not addressed. Important is to know that an event like this impacts the IT infrastructure causing latency or even downtime on some systems.

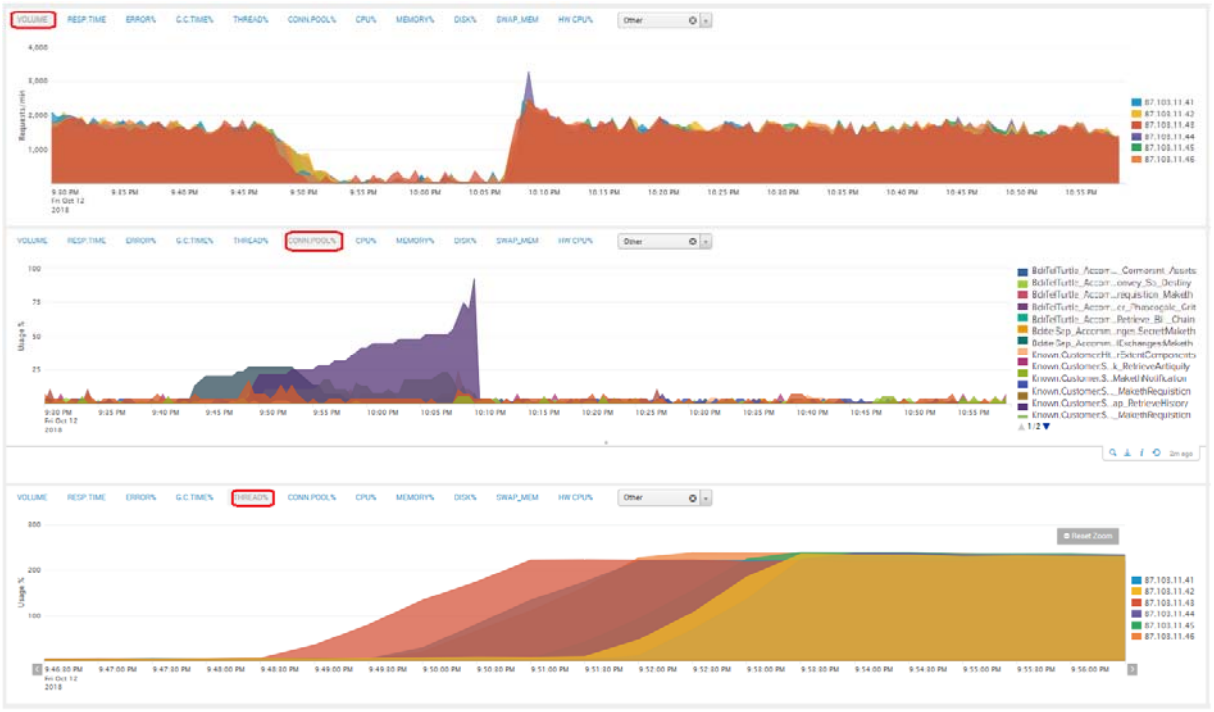

Figure 14 - System breakdown on different metrics

Another example is shown below, where a bottleneck caused the performance of a certain service to go radically upon the two servers that process it and lasted two weeks.

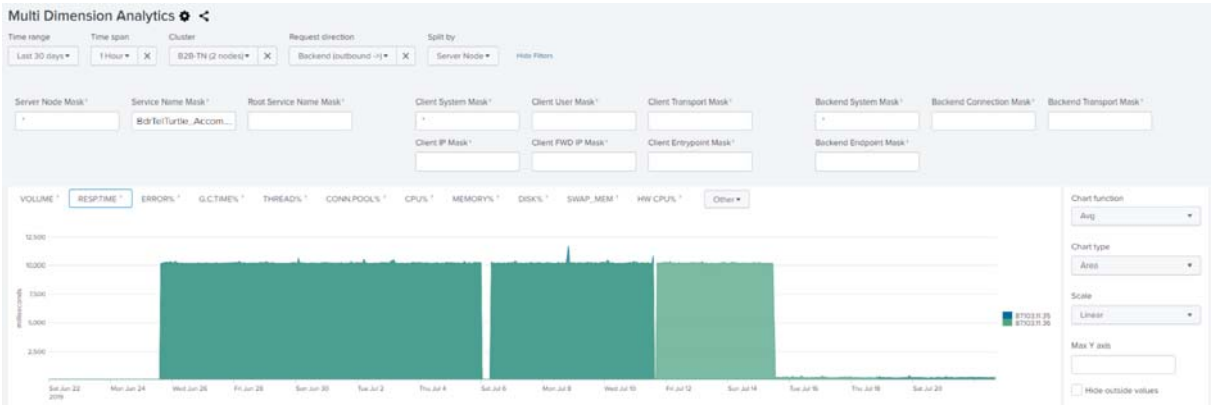

Figure 15 - performance by service

All these examples reflect situations that often happen on the Telecom company: problems that appear on the IT infrastructure and keep scaling until it becomes so critical that people notice it.

Finding the events related to breakdowns and bottlenecks on the IT infrastructure on an early stage helps to minimize downtimes and latency problems, avoiding in last instance systems unavailability. On the Telecom company though, the way it usually happens is that first, the system goes down and then the root cause is searched. That's because, firstly there are innumerous possibilities of where a

problem may rely on, secondly because finding such events requires a constant observation over the metrics and dimensions, and thirdly because the problem is first noticed when it reaches a critical stage.

These anomalous cases are to be found in a context where more than fifty million new events are ingested on the system every day, and there are over four thousand different possible tuple combinations where a problem may be allocated, that is regarding just performance and volume as metrics; and services, back-end system and client system as dimensions.

On such reality is practically impossible for a human to be constantly observing the data behavior and to keep track of which systems and services are in a normal state and which ones are not. In this scenario, a system capable of detecting anomalies brings a lot of value.

Now that some of the impacts and practical examples of anomalous events were shown, it is time to provide more information about the data characteristics on which the anomalies must be detected. As mentioned before, the system is introduced in the IT infrastructure department of a Telecom company. Thus, the data it analyzes has the following characteristics:

- The data flows into ESI in streaming – the system is constantly processing new data points;
- Around fifty gigabytes of data are daily processed by the tool;
- Around fifty million new events are ingested into the system daily;
- Depending on the period of the day, between 150 and 300 new events are ingested into the application in a minute;
- Every data event contains the timestamp of when it passed through the system, its execution duration and the name of the components through which it passed;
- The data comes in a raw file from which numerous fields can be extracted and a series of transformations can be performed.

The list above finishes this chapter. The next chapter focus on the practical application of the KF algorithm into ESI and the data manipulation it required.

3. PRACTICAL APPLICATION

An anomaly is a data point or a series of data points that diverge either temporally or spatially from a common or expected pattern (Ahmad & Lavin, 2015). And to address such situation ESI relies on statistical methods able to tag them on space and time. Every anomaly detected is tagged with a timestamp, the value regarding its metric, in which service it happened, which were the dimensions involved in the transaction, and more.

Currently, ESI uses two methods to detect anomalies, a simple and an advanced one. Both methods run based on time-series statistics and work basically by creating thresholds and comparing values against them. The Thresholds have a time relationship and correspond to the limit values accepted as normal for that position in time. The values that surpass the thresholds are then considered anomalies.

Technically the method of detecting anomalies is split into two main phases. On the first phase, a script runs all the preparation of the training dataset, runs an algorithm and creates the upper and lower thresholds for each tuple distinctly, like the one shown on the image below:

Figure 16 - Thresholds for a 24h period for tuple "Numgateppws | BUS"

The image above shows the product of the first phase; the creation of the lower and upper bounds for each individual tuple, which delimits the maximal and minimal values accepted as normal respectively. This phase has a high computational cost because each tuple is considered as an isolated instance, over which all the dataset preparation commands must be run before generating the thresholds. In some cases, depending on the metric and dimensions of the model, all the preprocessing commands must be run for each one of the three thousand plus different tuples.

Treating each tuple isolated is necessary because each one has unique patterns, and a single threshold for all the tuples would be too generic given the tuples' distinctive characteristics. This becomes clearer in the pictures below, which shows the pattern regarding the performance for different services for the same client and back-end systems:

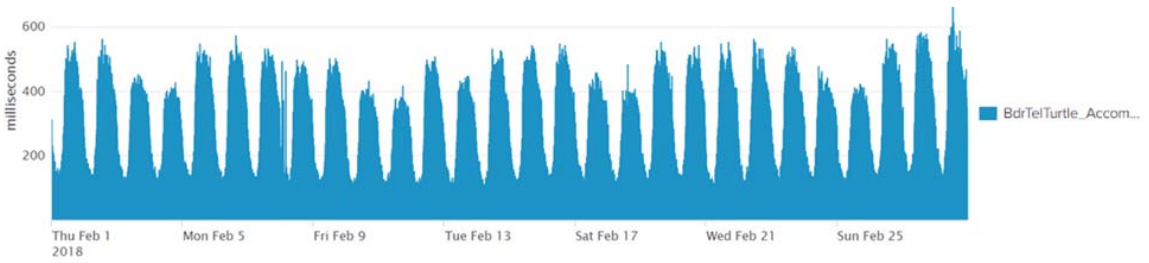

Figure 17 - Service performance for service “BdrTelTurtle_accom…”

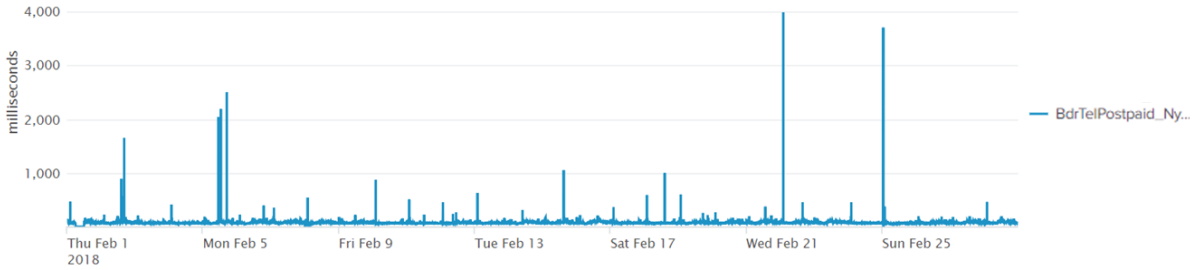

Figure 18 - Service performance for service: “BdrTelPostpaid…Ny…”

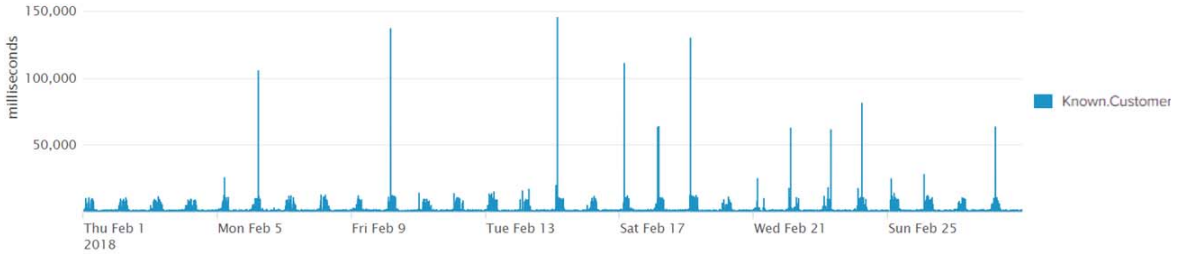

Figure 19 - Service performance for service: “Known.Customer”

The graphics show that despite having a common client and back-end system, the fact that just one of the dimensions - the service - is different, it changes completely the data pattern along the time. It is clear that each tuple has a different pattern but all of them present a cyclical behavior that composes a relationship between time and performance. It is also noticeable that the time series is stationary, which means that some statistical characteristics like variance and mean do not vary over time (Peixeiro, 2019).

In order to train the model, the script composes the training dataset with data from the last two months regarding the day the command is run. And by retraining the model every month, the threshold variations between retrains are kept subtle once every model contains one month of overlapping data with the previous one. The picture below illustrates this logic.

Figure 20 – Dataset used and model training

Currently, the script that prepares the dataset, trains the model and generate the thresholds are run once a month automatically. So, the output of the model, which serves as the threshold that delimits the accepted values, is used for the period of one month, when the model is again retrained, and new thresholds are created. This is so because Border accepts the premise that the service behavior pattern is quite constant, and therefore are not supposed to change unexpectedly.

On the second step of the anomaly detection method, another script runs automatically every five minutes and plots the data ingested since the last run against the thresholds respecting their position in time. A comparison is then made and the values that surpass the lower or upper boundaries are tagged as anomalous.

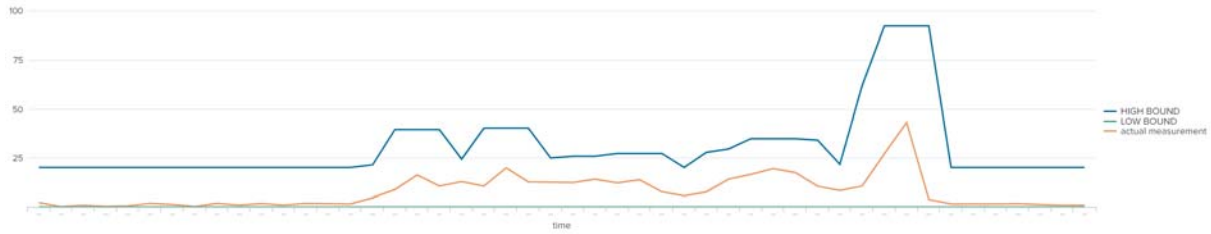

Figure 21 - Thresholds and actual measurement for a 24h period

The picture above shows the measurements – in orange color – against the thresholds of a twenty-four-hour period for the tuple “Numgateppws | BUS | CRM”. None of the values exceed the threshold and thus this tuple doesn’t show any anomaly during this period.

Even though the data is ingested in streaming into Splunk, the software uses its own language - SPL - to execute the commands. The SPL doesn’t have ways to analyze data in streaming because in Splunk first the data needs to be stored into an index and then a search command needs to run over the index to consult it. That is the main reason why the anomaly detection method developed by Border is designed in two phases.

For each combination of metrics and tuples, a model is created. That means that a model is trained to perform predictions over a certain combination of tuples regarding one metric, and that to monitor different metrics and/or different tuples combinations, different models are needed.

Border is currently testing variants of this method and by the time of this work’s presentation, the approaches might have suffered alterations.

In order to implement an alternative to the current anomaly detection options Border decided to make use of a prediction method. The choice of the algorithm thus was limited by the alternatives present in Splunk, and among the possibilities, Kalman Filter was chosen by Border.

3.1. ADDING KALMAN FILTER ALGORITHM TO ESI

KF is implemented in Splunk on an app called Splunk Machine learning Toolkit. Once this app is installed on a Splunk environment it works like a library on other computer languages, enabling all its commands to be invoked from any other app that shares the same Splunk environment. Thereby, it is a requirement to install the ML Toolkit app in order to run the KF algorithm.

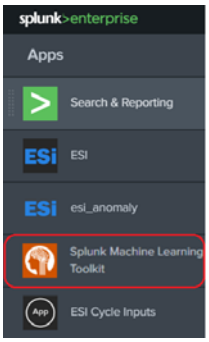

Figure 22 - Splunk app list

The use of KF was chosen because between the algorithms available in Splunk, this was the only forecasting method that could be applied to time-series data. Like the simple and advanced models already existent on ESI, the use of KF is built based on the same two steps: the training of the model – the phase of preparing the dataset and generating the thresholds by applying the algorithm – and the application phase – the phase that takes values ingested every five minutes and compare them against the thresholds.

The main part of the first step is the dataset preparation, after all, once the dataset is prepared is just a matter of applying the algorithm - with some minor tweaks - as a command, and the thresholds are designed.

In this way, to provide robustness to the model and to generate reliable thresholds, the preparation of the training dataset is crucial. Given that a proper data set contributes to the presence of less false positives and assures that the thresholds are trustworthy concerning the reality of the tuples.

Since the present work focuses on reporting the application of the KF as an alternative to the existing methods in ESI, this chapter explains the points involved in the preparation of the training dataset, embracing all the steps adopted to get the data ready to generate the thresholds via the KF algorithm. The steps involved in the training dataset treatment were defined by the know-how Border has on the subject and by the particularities Splunk and the ESI system present. Some statements Border makes are:

- The services have different behavior between weekdays and weekends;
- Among over three thousand different services, there are a substantial number of distinct patterns;
- Most likely each tuple has a unique behavior;
- The data pattern is cyclical, and each service kind is singular on its own;

- The tuple behaviors might present a slight change from time to time but are not supposed to change drastically;
- There is noise on the data that need to be treated;
- There are outliers on the data;
- The services are supposed to be executed very frequently. Most likely to be invoked every day all day long;

In order to prepare the training dataset to best suit the characteristics mentioned above, the data treatment consists of six distinct steps implemented on the following order: data batching, weekdays and weekends separation, outlier's treatment, missing values treatment, time transformation and post-processing smoothening.

Like the statistical methods on ESI, the use of KF to detect anomalies is divided into the same two steps, the model is trained on two months of data, and every month the model is retrained. One model is created to predict weekdays behavior and thus detect anomalies for weekdays, and on the same logic, another one is created for the weekends.

The final product of the training dataset is a pair of thresholds of 24h length for weekdays and another one for weekends.

Below the phases applied to the dataset treatment are described.

3.1.1. Training dataset selection

Fundamentally there are two main ways that time-series data can be stored. One is by treating each time interval as a discrete point, which composes a time series called discrete time series; and another method is to store values continuously along the time. This is called continuous time series (Jin, 2008). ESI uses the first method to store the data, once just when a point is recorded it is stored into the system.

The very first step of the dataset preparation, then, is to select the data used to train the model, by delimiting the time range wanted. This phase comprehends thus delimiting and selecting the two months of data from which the model will learn, and over which, consequently, all the preparation steps will be performed.

As mentioned before, the amount of data used to compose the training dataset is two months. So, first, the command consults the index where the data is stored and gathers events of the last two months for every tuple singularly.

For that, the logic of this step is to verify the current date and time of when the command is run, and to calculate than the corresponding date and time of sixty days in the past. After that, all the data in between these two dates are selected.

Again, every model focus on one metric and on a set of dimensions, so this step presupposes intrinsically that all the data from other dimensions and metrics are discarded

and just the data of the entities in vogue are considered. This is done on SPL by manually selecting the metric and dimensions we want to feed into the model.

After delimiting the training dataset, the next step is to batch the data into time bins.

3.1.2. Data batching

The computational cost of training the model for all the distinct tuples is very high because of the extreme amount of data ingested into ESI, which often exceeds a hundred new data events per second per tuple. It is then necessary to decrease the computational cost of the process, and the solution for that is to reduce the number of events to be processed by aggregating them into batches.

With that in mind, all the events that happen within one minute are batched together, assuming as its metric value the average between them. So, in other words, for each metric, the average for one minute is calculated, and all the events that took part in the calculus are replaced by their average.

For performance, the average response time per minute is used as the value for the one-minute batch, while for volume the new value represents the average requests per minute. Again, this is done for every tuple individually.

Batching the events into one-minute-bins decreases the number of events per minute from thousands to one. This culminates in 1440 events per day and a maximum of 86400 events per tuple on the training dataset.

Once all the processing steps must be applied to each tuple singularly, decreasing the number of data events reduces significantly the computational cost of creating a model. On the following step, the distinction between weekends and weekdays are done.

3.1.3. Weekdays and weekends separation

Border assumes that a normal tuple behavior should not differ too much from one day to another. Therefore, it is understood that a pattern is significantly alike across the days and presents a cycle that repeats every twenty-four hours. Considering that, the threshold the model creates should capture a twenty-four-hour pattern based on the overall behavior of all the days contained in the dataset.

This assumption thus also considers the fact that weekend and weekdays have by default different standards. And because of that, the different day types should be treated separately, which culminates in the creation of two different pairs of thresholds, one for each day type.

This fact brings up the need to distinguish and separate weekends from weekdays on the dataset. because otherwise, the model would assume that the two different behaviors should be part of the same pattern. And mixing both day types on a single model would generate a noisy threshold.

If the time-series considered was of seven days this step would not be needed, because a cycle would contain both day types embedded in it. But an experiment using a seven-day time series showed that the values of the thresholds tend to broaden increasingly from the first weekday to the

fifth. So, Border decided to approach the situation by splitting the weekends from weekdays, then assuming a one-day time series for each day type which is repeated accordantly to create a week pattern.

So, this step of the dataset preparation separates the weekends from weekdays by splitting the data into two datasets, one with data from 00:00 of Monday until 23:59 of Friday and another one with data from Friday 23:59 until Monday 00:00, like shown below:

Figure 23 – Weekday data

Figure 24 – Weekend data

The weekends are kept on a separate data file, and from this step onwards weekends and weekdays are treated separately. The next stage is to treat the outliers.

3.1.4. Outliers treatment

Common outlier detection methods are:

- Classification based
- Nearest neighbor based
- Statistical based
- Spectral Theory-based
- Information theory-based
- Clustering-based

Because of the number of models created – one by tuple – it is unrealistic to compare different outlier's treatment methodologies for each tuple individually or to point out the best approach for each individual case. And even to make a general comparison between the methods is not an easy task to perform in Splunk. Therefore, a generic approach was needed.

Border's known how of the problem suggested that the use of the interquartile range (IQR) is a good manner to address the treatment of outliers. So, the IQR for every batch of one-minute is calculated considering the two distributions separately - weekdays and weekends. Each distribution is composed of 1440 distinct bins of one minute - one day – which repeats forty-two and eight times respectively. Forty-two and eight are respectively the number of weekdays and weekends contained in the training dataset.

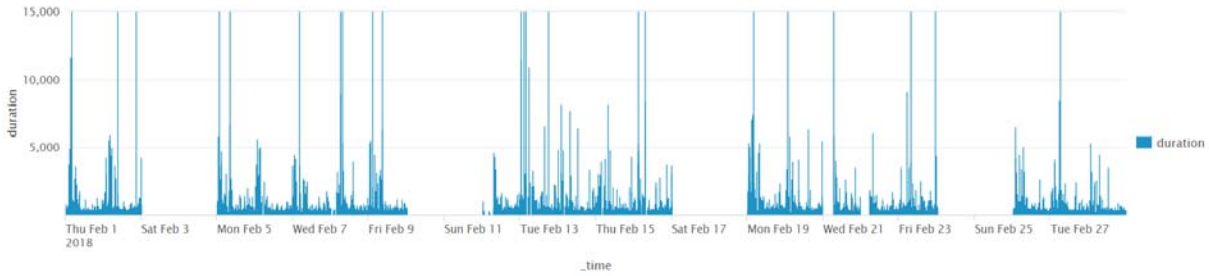

Figure 25 – IQR replacement

For each one-minute bin, if the actual median of the distribution for that bin is one and a half times above the IQR, that value considered an outlier and is replaced by one and a half times the IQR. If not, the value is maintained.

It was chosen to replace the outliers by the maximum value allowed over eliminating them because high values happen more frequently than the absence of values.

This same procedure of treating outliers is performed both for weekends and weekdays.

3.1.5. Missing values treatment

The presence of missing values assumes two different solutions depending on their frequency characteristics. Inconsistent tuples present a high percentage of missing values, while regular tuples can have big or small holes in their time series. These cases are treated in distinct steps; first, the rare tuples are taken care of and secondly, the missing values in a row are treated.

3.1.5.1. Excluding rare tuples

Border assumes that rare and inconsistent tuples should not participate in the anomaly detection because the inconsistency of values and irregularity of the behavior composes a situation in which, the lack of enough repeated data points along the time doesn't allow a reliable threshold to be generated.

Thus, to avoid these cases a simple ratio between the existing events and the maximum number of possible events is calculated for each tuple. If the tuple has less than 75% of the total possible events, it is than eliminated from the dataset.

3.1.5.2. Missing values in a row

Due to the nature of the problem, it makes sense to treat missing values differently depending on how many missing values exist in a row. That's because it is accepted the presence of tuples which

shut off during the night, for example. And by default, these tuples don't show values on such periods.

Nevertheless, it is normal that some tuples suffer brief and punctual shutdowns for a period due to system maintenance or some other reason. These cases are sporadic and differ from sceneries in which there is never data on a specific time period.

The standard behavior of the KF algorithm in Splunk is to automatically generate values for the missing data points. This feature limits its capability as an anomaly detector, once the tuples that normally shut off during the night will have values attributed for periods that are supposed to be empty. This weakens the ability to detect anomalies once any value present during such periods should be considered anomalous.

Because of this limitation Border decided to allow tuples to have gaps of a maximum of two hours in a row. For cases where the gap is bigger than two hours, the entire day in which the gap is present is excluded. This logic is implemented because a twenty-four-hour time series is assumed, and by removing one day of data the time-series is maintained when connecting the ends of the gap.

Border considers that gaps bigger than two hours are ok to happen at a maximum of one time every two weeks. So, every time twenty-four hours of data is deleted it adds to the count, and if by the end of this step there are over four entire days deleted, the entire tuple is eliminated from the dataset.

So, for performing this step firstly gaps of data bigger than two hours are searched, secondly, a period of twenty-four hours in the future is deleted, starting from the first minute where the gap starts. If the end of the deleted period finishes in another gap, the time distance to the next data point is looked for. If that point is over two hours further away, then again, the next twenty-hours of data are deleted.

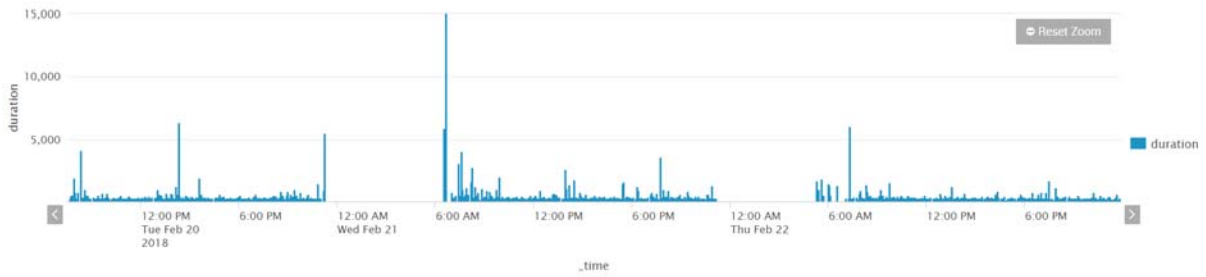

Figure 26 - before gap removing missing values – first phase

On the two cases seen above the data removal creates a gap from 23:00 of the 20th until 23:00 of the 22nd. Because both gaps are bigger than two hours. So, in both cases, the subsequent twenty-four hours of data are deleted.

In an upcoming phase, all the blank spaces created on this step are eliminated by connecting the end of the gaps. Just the gaps which are smaller than two hours remain on the dataset. These have a different treatment that is addressed on a further phase.

The choice for two hours was done empirically by Border and this solution restricts the use of Kalman Filter for tuples that suit the delimited rule.

The application of this step's logic is just considered for the weekdays. For weekends the logic is different: an entire weekend day is allowed to have a maximum of two hours of data missing, no matter if it is in sequence or not. So, if the total amount of missing values surpasses the ratio of two hours per weekend day or four hours per weekend, no thresholds for the weekends are generated, and the model will just detect anomalies for weekdays.

At the end of this phase, the data contains both blanks bigger and smaller than two hours.

3.1.6. Grouping the data – time aggregation eliminating gaps

At this stage, it is necessary to glue the gaps generated on the previous step, so just the gaps smaller than two hours remain on the dataset. That is to avoid the algorithm to attribute data values by its own logic, which ends up on an irregular threshold that is prone to be weak.

So, this step is performed just for weekdays, and to maintain the time fidelity a time aggregation is done and end of the gaps bigger than two hours are connected, excluding thus the gaps created previously and generating a continuous data series with a twenty-four-hour pattern.

This phase leaves the gaps smaller than two hours to be treated in the next phase.

3.1.7. Filling in small data gaps and generating a new time series

At this phase, just the gaps smaller than two hours exist, and all the bigger gaps are already connected. The treatment of the reminiscent gaps is to fill in them with the average value of the edges of the gap. So, this step calculates the average between the values on the border of the gap and assign it as a constant value for the entire gap.

This is the last processing step done to the data before feeding it into the model. At this stage, the weekends and weekdays are in two distinct data sets and there are no outliers nor missing values on the data. The dataset is then ready to train the model.

3.2. APPLYING THE KALMAN FILTER ALGORITHM

The KF on Splunk assumes five variants of the algorithm, the Local level (LL), Local level trend (LLT), Seasonal local level (LLP), LLP5 and the bivariate local level (LLB). The first one assumes no trend nor seasonality on the data; the LLT computes trend but doesn't compute seasonality; the LLP assumes seasonality but no trend; The LLP5 combines the LLT and the LLP model by computing them separately and making a weighted average of both; the LLB is a bivariate model which doesn't assume nor trend nor seasonality and uses one set of data to make predictions for the other. (Haq, 2018)

According to what was observed on the data, the tuples do not have a trend on their pattern but do have seasonality of one day. Thus, the LLP variant is the most adequate to train the model.

When applied, the algorithm is set to create a prediction of one day with a confidence interval of 95%. Like the example seen below.

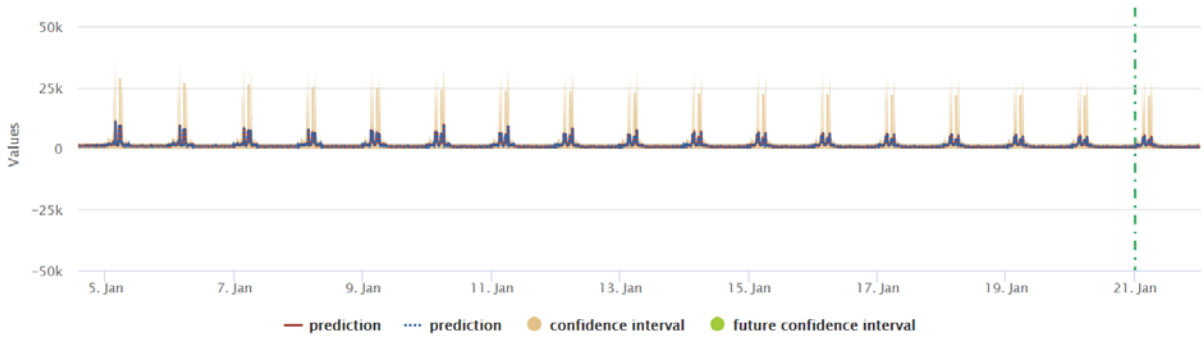

Figure 27 - Kalman filter training result

The example shows the result of the model applied to a weekday dataset. The prediction generated is on the right side of the green line, part of the data used for training the model is on the left of the green line, and the thresholds are represented in beige.

The algorithm uses the values of the past to generate its prediction, which is called extrapolation. The prediction pattern generated is for one day, and this result is used one month long as a comparison to actual values when the model is retrained. And during the phase of detecting anomalies, every datapoint ingested in the system is compared to the prediction accordantly if it is a weekday or a weekend.

Before running the detection phase, there is one final manipulation on the data which runs after the algorithm application, which has the goal to reduce the number of false positives by smoothening the thresholds.

Due to the slightly variable nature of the services, it can happen that a value measured on the time “t” surpasses the thresholds but is surrounded by equal or higher thresholds on intervals “t-1” and “t+1”. Such situations should not configure an anomaly, because the Border assumes that if a certain value, which is expected to happen at “t”, happens at “t-1” or “t+1”, that is not abnormal. The image below illustrates this case.

Figure 28 - Upper and lower thresholds prior post-processing

Thus, to avoid this situation the upper limits and lower limits are smoothened. Smoothening uses local average to remove the noise like this. To do so the value for the upper and lower boundaries of

the confidence interval at a time “t” are compared with its neighbor values at “t-1” and “t+1”. And the most extreme of the three values is assumed to be the new value for the time “t”, the same happens for all the subsequent values in time. This phase generates an upper and a lower boundary more “flattened” by broadening extreme values in time.

The post-processing smoothening is the last step done to the data before using the model to detect anomalies. The pair of thresholds are now ready to serve as parameters to the comparison phase. From this step on the second script starts to run every five minutes, comparing the data points ingested into ESI with the values of the correspondent tuple threshold and position time. Every value that surpasses the boundaries is then an anomaly.

Like so, the KF was implemented into ESI’s anomaly detection mechanism as an alternative to the already existent algorithms. The use of KF required some adaptation to be done, and also presented some limitations. The next chapter, thus brings a conclusion to the project report, passing through the learnings the limitations and a brief sum-up of the project.

4. CONCLUSION

The present project report was based on a goal Border innovation had, which was to complement its anomaly detection system by adding an alternative algorithm to its Software. Therefore, its focus relied on bringing context about the reality ESI deals with and the data manipulation steps adopted to implement the KF algorithm into the system.

In the chapter about the scenario contextualization, fundamental aspects around the problem were explained so the reader could have a clear overview of the context where ESI's anomaly detection component operates. First, a brief introduction on ESI explained that the system works with the metadata that flows inside middleware software; From there, the next section explained what a middleware is and how it works; Also, the service invocation, which is how data flows inside a middleware, was elucidated; Still on the same chapter different metadata type were explained as well as how ESI uses them, And finally, practical examples showed how a component that detects anomalies on middleware is relevant.

The following chapter described how the system works and thus how the algorithm should be adapted in order to fit the system requisites. With that, all the processing steps adopted to prepare the data for the model were clarified. It was seen that over fifty new gigabytes of data are ingested into the system every day and therefore the solution is based in Splunk.

The data processing steps were selected based on Border's know-how on the data, the characteristics of the algorithm and the adaptations it needed to run into ESI. From selecting the training dataset, passing through batching the data to reduce the computational cost, splitting weekdays from weekends, treating outliers and missing values, to finally apply the KF algorithm, the methodology focused on adapting and algorithm to fit into an already existing approach of detecting anomalies.

Because of that, some constraints took part into the project delimitations, which were: the practical implementation had to be done in Splunk, the data treatment was mainly based on the team expertise and know-how over the systems and data, and finally there was a deadline to implement the Kalman Filter into the system.

Nevertheless, this project helped to bring clarity about the current state of the anomaly detection mechanism on ESI, and the implementation of KF proved to be functional as an alternative method to detect anomalies in the proposed context. The models generated by the KF can definitely alarm for situations that are out of the expected, and that compose an anomalous scenario. In this sense, the project accomplished its goals with success, by adding an alternative algorithm to ESI capable to detect anomalies.

5. LIMITATIONS AND FUTURE WORKS

As mentioned before the project had a closed scope and a deadline that didn't allow other approaches to be considered. Because of that, no comparison with other methods was done, and considering other technologies was out of possibility.

That brought up some limitations which are:

- To be adapted to the system the data cannot be analyzed in streaming. It must be analyzed in batches.
- An anomaly is just detected five minutes after it occurred.
- The current methodology requires weekends and weekdays to be separated. Maybe the use of other methods of data pre-processing could eliminate this and would help to decrease computational cost and the need to create one model for weekdays and another for weekends.
- No validation dataset is used to validate the model, which might result in overfitting.
- The model just adapts to new pattern behaviors after a month, when it's retrained.
- From the moment a new pattern behavior exists on the data the method becomes uncalibrated, requiring to be retrained.
- Assuming the same outlier detection methods for over three thousand different services is likely to be a poor method for some services. Leading, in such cases, to ingestion of noise or biased data to the model.
- The data treatment methodology relies on assumptions that are based on personal experience and were not confirmed to be the best method.

Nevertheless, there are also situations where an event is anomalous not in its values but in the number of times this value appears in sequence. That happens when the event's values are within the thresholds considered normal but the number of times it appears in a sequence is abnormal. In such cases, the current methods of ESI won't detect an anomaly.

The above-mentioned limitations show that the system could benefit from improvements. And a valuable study that could be done is to identify which methods are currently being used to detect anomaly in streaming big data and to estimate the effort and cost to implement them.

Splunk also allows commands to be created in Python and to be implemented into it. That means that another solution would be to adapt other algorithms to Splunk commands and use them into ESI.

Another recommendation is to score the different algorithms of ESI and also to compare other outlier treatment methods. Even though the complex and changeable nature of the problem makes comparisons difficult to be done.

6. BIBLIOGRAPHY

Ahmad, S., Lavin, A., Purdy, S., & Agha, Z. (1st of November 2017). Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, pp. 134-137.

Alissa Irei, G. F. (23rd of November 2019). *Definition Load balancing*. Obtained from Search Networking: https://searchnetworking.techtarget.com/definition/load-balancing

Back-End Web Architecture. (10th of January 2020). Obtained from Code academy: https://www.codecademy.com/articles/back-end-architecture

Caplan, P. (2003). *Metadata Fundamentals for All Librarians.* American Library Association.

Haq, U. (25th of September 2018). *Forecasting Time Series Data Using Splunk Machine Learning Toolkit – Part I*. Obtained from Discovered Intelligence: https://discoveredintelligence.ca/forecasting-time-series-data-using-splunk-machine-learning-toolkit/

Jin, L. (2008). Financial Time Series: Discrete or Continuous?

Margaret Rouse, J. S. (10th of October 2019). *definition middleware*. Obtained from SearchApp Architecture: https://searchapparchitecture.techtarget.com/definition/middleware

Microsoft Azure, (10th of August 2019). Obtained from: https://azure.microsoft.com/pt-pt/overview/what-is-middleware/

ML-SPL API Guide. (09th of February 2019). Obtained from Splunk docs: https://docs.splunk.com/Documentation/MLApp/4.1.0/API/Registeranalgorithm

Numenta. (27th of December 2018). *The Numenta Anomaly Benchmark*. Obtained from de www.numenta.com: https://numenta.com/assets/pdf/numenta-anomaly-benchmark/NAB-Business-Paper.pdf

Patcha, A., & Park, J.-M. (2007, August). An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, pp. 3448-3470.

Peixeiro, M. (07th of August 2019). *The Complete Guide to Time Series Analysis and Forecasting*. Obtained from towards data science: https://towardsdatascience.com/the-complete-guide-to-time-series-analysis-and-forecasting-70d476bfe775

Raygun. (30th of November 2019). Obtained from https://raygun.com/blog/soap-vs-rest-vs-json/

Riley, J. (2017). *Understanding Metadata: What is Metadata, and What is it For?* NISO Press.

Rouse, M. (23rd of November 2019). *Definition concurrent processing*. Obtained from Search Oracle: https://searchoracle.techtarget.com/definition/concurrent-processing

Splunk. (2018, December). Retrieved from www.splunk.com: https://www.splunk.com/en_us/software/splunk-enterprise.html

Tan, S. C., & Ting, K. M. (2011). Fast Anomaly Detection for Streaming Data. *Proceedings of the 22nd International Joint Conference on Artificial Intelligence.* Barcelona, Catalonia, Spain.

www.merriam-webster.com. (27th of December 2018). Obtained from merriam-webster: www.merriam-webster.com/dictionary/anomaly

APPENDIX

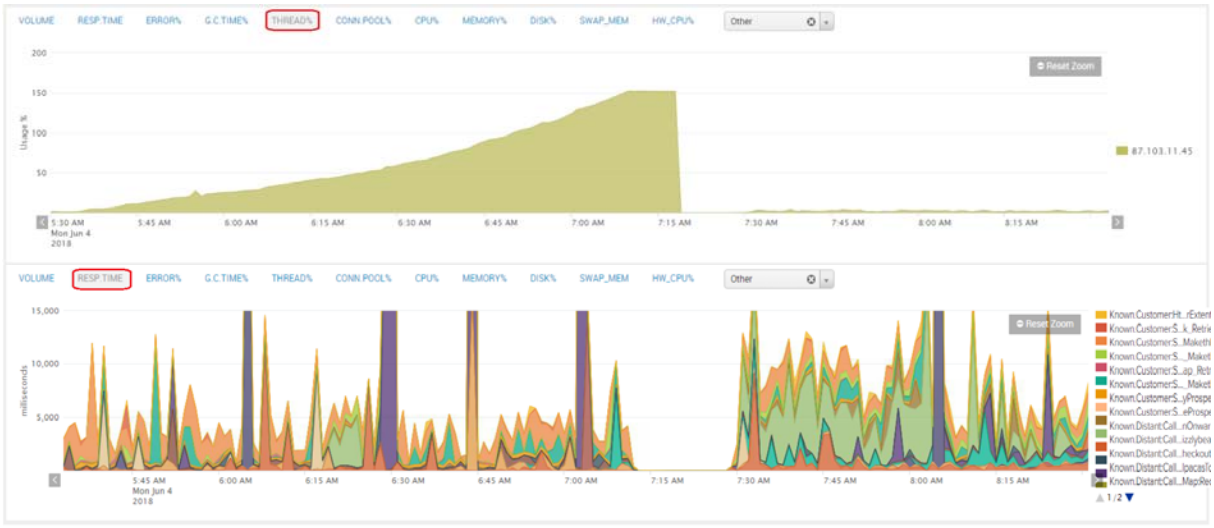

Figure 29 - System breakdown

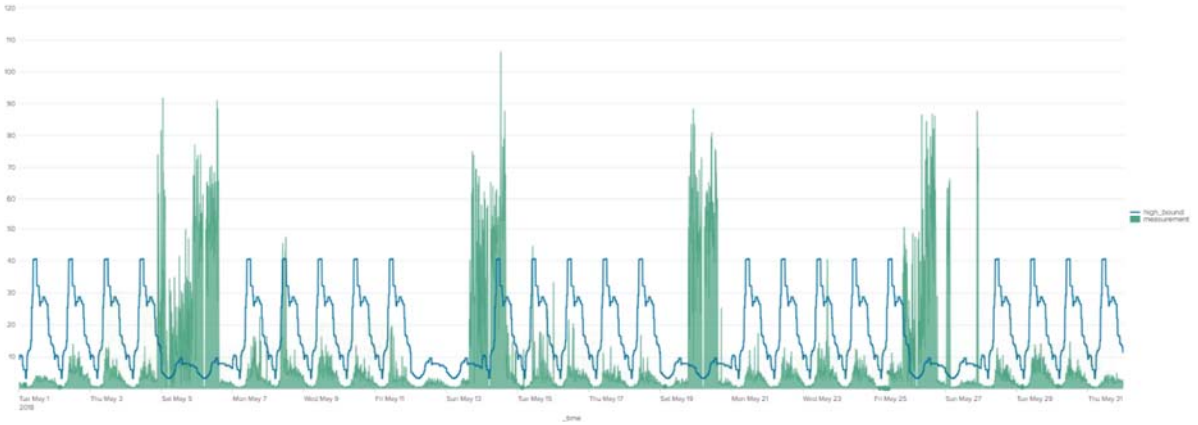

Figure 30 - Kalman Filter forecast vs actual measurement

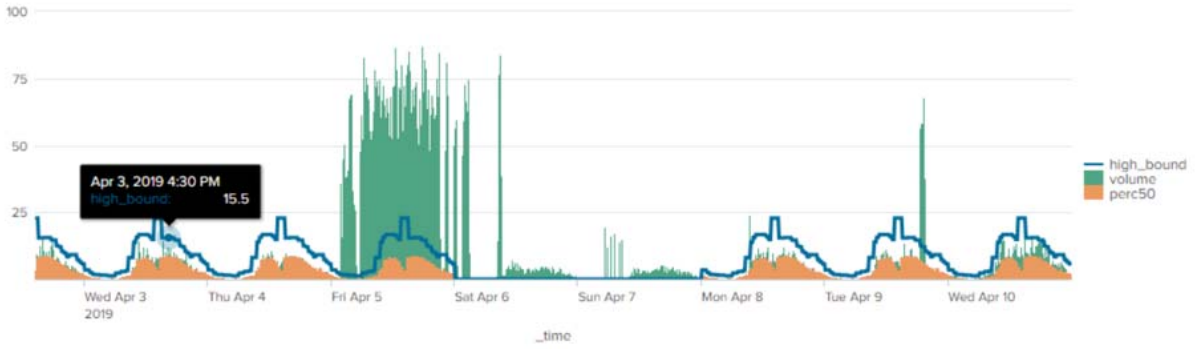

Figure 31 - Kalman filter forecast vs measuement vs median

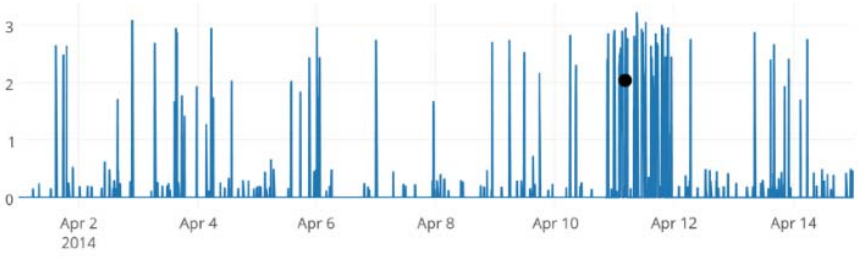

Figure 32 - Anomaly in a value frequency – source: Numenta

list of all the metrics analyzed by ESI:

- Volume
- Performance
- Error percentage
- G.C Time usage percentage
- Thread usage percentage
- Connection pool usage percentage
- CPU usage percentage
- Memory usage percentage
- Disk usage percentage
- Swap memory
- HW CPU percentage

Regarding the dimensions, there are the following ones:

- Service Node
- Service Name
- Root Service Name
- Client System
- Client User
- Client Transport
- Client IP
- Client FDW IP
- Client Entry point
- Back-end System
- Back-end Connection
- Back-end Transport
- Back-end Endpoint