Fachhochschule Aachen, Campus Jülich

Fachbereich 09 - Medizintechnik und Technomathematik
Studiengang Angewandte Mathematik und Informatik

Seminar Thesis

Statistical analysis of log occurrences generated in Cloud Environments

Autor:
Nils Neumann, 3280760

Ausbildungsbetrieb:
Ericsson GmbH

Aachen, December 19, 2022

Confidentiality Clause

This thesis contains confidential data of Ericsson GmbH. This work may only be made available to the first and second reviewers and authorized members of the board of examiners. Any publication and duplication of this thesis - even in part - is prohibited. An inspection of this work by third parties requires the expressed permission of the author and Ericsson GmbH.

Prof. Dr. Philipp Rohde

M.Sc. Hannah Hechenrieder

Aachen, December 2022

Statutory Declaration

I hereby assure that this thesis with the title

"Statistical analysis of log occurrences generated in Cloud Environments"

has been produced and written on my own and has not been part of any other study or examination performance. No other means, sources or resources have been used, other than the listed ones. Every part of my work that contains quotations is marked accordingly and can be found in the bibliography.
I vow to keep a copy of this thesis for five years and to hand it out on request by the examination office of the faculty "Medizintechnik und Technomathematik".

Location, Date	Nils Neumann

This thesis was supervised by

Prof. Dr. Philipp Rohde	M.Sc. Hannah Hechenrieder

Aachen, December 2022

Abstract

As software systems tend to get very big nowadays they produce a lot of logging data. This logging data can be very challenging for a developer to understand. But understanding logging data brings essential information when it comes to debugging CI pipelines running in a cloud environment because otherwise, it can take up much time for a developer. This can lead to big costs for the production because some bugs and errors can have a big impact on the customers.

Ericsson's tool Lexicon helps troubleshooters to get a better overview of this big amount of logs. It shows the users whose logs tend to be abnormal. This helps to find the error much quicker. A log is abnormal if Lexicon has seen it only a few times or not at all in its training sets. Because this rating does not detect abnormal frequencies of logs, this thesis searches for possibilities to detect these abnormal frequencies.

Taking a look at logs whose frequency differs much from the average log frequency, which is at two, shows that these logs are minor warnings without a big impact.

By looking at which logs do not occur in unsuccessful runs, we see some important logs like health checks, that could have led to the failure of unsuccessful runs by missing in them.

Logs that only occur in unsuccessful runs have a higher chance of being related to errors. But there are a lot of logs that are not related to error and are just normal logs. So abnormal frequencies cannot classify abnormal logs on their own but can contribute to the existing abnormality rating by giving another factor to consider.

Contents

1 Introduction

1.1 Motivation

Digitization has led to an ever-bigger demand for software solutions. With it also comes a huge increase in the complexity and size of software projects as the number of requirements for software rises quickly. This brings up various challenges for the developers, system administrators, and support teams because with a bigger size of software projects the potential for errors also rises. With the growing complexity of software systems, it gets increasingly challenging to find the causes of these errors. This can be very detrimental as it can have a direct influence on the customer experience. The US National Institute of Standards and Technology estimates that $60 billion are lost due to problems caused by software errors.[Cun09] So there is a big demand to keep the time to find and solve these errors as short as possible.

This is where logging comes into place. Logging is a very common programming practice for developers to understand what happens in their software. [YPZ12] If logging is used correctly it can be very helpful to keep track of errors and to find the error's source quickly.

Therefore, analyzing logs plays a big role when it comes to solving errors in a program. With the increasing size of software systems, they also produce a lot more logs which can make analyzing them very challenging.

This problem can be solved by using log analyzing tools like Ericsson's Lexicon. Lexicon learns from previous runs by CI pipelines with various methods of machine learning and mathematics. CI pipelines get explained further in Chapter 2.1. After learning Lexicon

gives its users a fast overview of their logging data to find the program's problem faster. To do so it collects all logs and clusters them into groups. Then Lexicon gives each log type an abnormality rating based on the number of runs it occurred in the training. This rating does not include if a log type occurs with an abnormal frequency. A log type could occur much more often or much less in a run than it used to. This thesis analyses the frequencies of log types of a run to show if these values are relevant for the abnormality rating.

2 Background

2.1 Cloud Environments

Cloud environments enable the developer to separate an application from its underlying physical equipment by outsourcing it into a virtual environment.[Gee22]. This process of outsourcing the software into a cloud environment is also called virtualization. It leads to an enhancement that includes better viability through faster response times, improved storage, easier scalability, and decreased computing times. It also opens up the possibility to use multiple machines that share one physical instance of the resources, which can make organizing big networks for organizations a lot easier. [Bah22]
To help define cloud computing, the National Institute of Standards and Technology (NIST) has collected five key requirements that a cloud environment needs to provide. These requirements are:

- On-demand self-service, which means that clients can use the environment by themselves without needing an administrator
- Broad network access, which means that resources are available with a basic network connection
- Resource pooling and multi-tenancy, which means that resources get pooled in the environment and provided for each user separately
- Rapid scalability and elasticity, which means that resources can be changed in real-time to satisfy client's needs

- Measured service, which means that the environment gets controlled and optimized continuously

[Sar21][MG11]

CI pipelines are often used to test processes in cloud environments. CI stands for continuous integration. These pipelines aim at automating the building process and discovering errors early.[Pen21] To do so they build and test changed software packages and merge them to the main project through successful tests. As these pipelines usually are used for developing big software systems, their runs often have a huge output of log data which can be very complex.

2.2 Practice of logging

Logging is the process of tracking events that occur when software is running. [Gee21] The purpose of logging is to record what happened in the program, which is especially important when the program crashes for an unknown reason. If a crash happens and there was no logging done the chances for the developer to detect the cause of the crash are very poor. [Gee21]

Logging is done by writing text messages into a secondary interface. The generated messages are called logs or log lines. Secondary interfaces are for example files, web services, or even emails. Logging into so-called log files is the most common secondary interface. [Ebe14]

Usually, log lines contain a timestamp, the type of the log, and the log message. The timestamp indicates when the log was written. The type is a way to effectively categorize the log line. Typical categories are INFO, WARNING, ERROR, FATAL, or

DEBUG.[Ebe14]

Examples of logs are:

2009-06-05 08:15:23 [INFO] User credentials entered

2009-06-05 08:15:23 [WARN] Incorrect PIN

This example demonstrates the software of an Automated Teller Machine (Cashpoint). [Ebe14] The first log line gives the information that a user tried to log into the service. The second line indicates that the attempt of user to log in failed because the user entered the wrong pin. This is classified as WARNING because it is something that is not necessarily supposed to happen. It is not classified as ERROR, because it is something that is expected to happen and will not lead to a crash of the program.

Despite these measures taken to increase the readability of log files, they can get very large and complicated. A very big size of log files also brings up many logs that are classified as error or warning, that are not relevant to the whole system. That makes it really challenging for a software developer to find the cause of an error when the application crashes. The complexity of finding the error can be even higher when logs are not written as they should be. That includes for example logging redundant information or not logging enough information.[Zha15]

This can lead to serious costs for a production when it comes to debugging a crashed system.

2.3 Lexicon

Lexicon is a log analysis tool to detect anomalies in log files and to determine the root cause of a failure in a system using various machine learning techniques. It tries to reduce debugging costs by supporting the developers during troubleshooting. To do so it aims to show the logs more structured and to locate the root cause of a failure.

Its most common use cases are CI pipelines as these have a huge amount of log data that can be very complicated to understand for a developer without any help of software.

Lexicon learns from past runs of CI pipelines that are already labeled as successful or unsuccessful and creates a model for future runs. To process the log data from the training runs, it goes through several stages within Lexicon. For the context of this thesis, the most important stages are the log parser and the log type mapper.

The log parser filters the variable part from the log lines. It replaces the variable parts such as timestamps, version numbers, IP addresses, port numbers, and terms that only occur in a single run with placeholders. The single-run terms are taken into account because it is likely to be an individual id. Figures 1 and 2 below show an example of how the log parser works.

renewTime: 2022-08-19T14:29:27.301108+00:00 transitions: 0 ttl: 30 Data ==== BinaryData ==== Events: none

"timestamp":"2022-09-01T15:09:22.866Z","version":"1.0.0","message":"Checking request :"/actuator/prometheus","correlation_id":"ab7ce0c5b019b485", "severity":"debug"

"timestamp":"2022-09-01T15:09:23.233Z","version":"1.0.1","message":"Checking request:'/actuator/prometheus/info","correlation_id":"9af4d48926714ad7", "severity":"debug"

Figure 1: Example of logs before log type mapping

renewtime __timestamp__ transitions __num__ ttl __num__ data binarydata events none

timestamp __timestamp__ version __version__ message checking request __path__ correlation_id __generated_word__ severity debug

timestamp __timestamp__ version __version__ message checking request __path__ correlation_id __generated_word__ severity debug

Figure 2: Example of logs after log type mapping

Figure 1 shows three different logs in their initial form. The variable parts that are relevant to the log parser are highlighted in different colors. Figure 2 shows the same

logs after they were parsed by the log parser. The variable parts are substituted with different placeholders that indicate what they are substituting. The log parser also removes symbols from the log. These cleaned logs then are used by the log type mapper to create log types.

The log type mapper collects logs from various runs and clusters these logs into their log types. A log type groups similar logs together as these log's messages most likely have the same meaning. They are grouped based on the similarity of their cleaned message. For example, in figures 1 and 2 the second and third logs would be grouped into one log type. Their similarity to one another is very high because, besides the variable parts that were replaced by the log parser, they are identical. Afterward, the log type mapper calculates each log type's abnormality. The abnormality takes into account in how many runs the log type occurs. A log type gets classified as very abnormal if it is unknown to Lexicon because it did not occur before.

By rating the log types with this method, abnormalities, like log types not occurring at all or much more than they used to, are overseen.

The abnormality rating can be improved by taking more factors into account. This thesis aims to provide background knowledge if taking not only into account in how many runs a log type occurs but also considering how often a log type occurs in each run can help to improve this process. To establish that, a dataset of logs was analyzed with various methods of descriptive statistics.

2.4 Methods of Descriptive Statistics

> "Descriptive statistics, in short, help describe and understand the features of a specific data set by giving short summaries about the sample and measures of the data."[Hay22]

Descriptive Statistics enables summarising a given dataset via brief informational coefficients. Factors to be measured are frequency, central tendency, dispersion/variation, and position.[Yel18] These factors are measured in:

- The frequency of data can be described with the distribution. It refers to the number of occurrences of a value in the dataset and is usually displayed in a histogram.
- The mean is used to describe the central tendency of the dataset. It calculated the average value of the dataset by summarizing each value of the dataset and dividing it by the total number of data points.
- To describe the dispersion of the dataset the variance, standard deviation and the range can be used. The range describes the distance between the highest and lowest point of the dataset. The variance and the standard deviation both describe how much the data is spread from the mean. The standard deviation is the root of the variance.
- The position of the values of the dataset can be determined via quartiles and the median. The median describes the value of the data point that is positioned in the middle of the dataset. The quartiles represent, similar to the median, the points that are positioned at 25 and 75 percent of the dataset.

Sometimes it describing a big dataset with these descriptive values is not enough to get a clear interpretation of it. In that case, there are plenty of methods to visualize the dataset's numbers:

- Bar charts are used to compare a small amount of data by organizing it in separate bars on the graph. By comparing the size of the bars you can easily see the proportion of the data points.
- A pie chart is an easy way to visualize percentages. A whole circle represents 100% and gets split up in the pieces regarding their part of the dataset.
- Histograms organize data in ranges and show the frequency of data occurring in these ranges.
- With scatter plots two datasets can be easily compared. There each axis stands for one dataset. The data points are visualized as dots and their position on the graph tells the relationship between the datasets.
- Boxplots display the distribution of the data. There is a box whose edges represent the upper and lower quantiles. In the box, there is the median. There are two whiskers going out of the box. They either represent the minimal and maximal value of the dataset or values within 1.5 times the interquartile range. The interquartile range represents the distance of both quartiles. So this type of whiskers lays 1.5 times the interquartile range apart from the quartiles. The boxplots in this thesis use this type of whiskers.

[Stu]

3 Analysis

3.1 Inspection of Dataset

For the analysis, I received a parsed dataset from Lexicon with 61 runs. Each run contains a true or false value, that shows if the run was successful or not, and a list of its logs. One log contains its id, a timestamp, its message, a list with a cleaned message, and information about the log type. This information contains the log type's id, in how many successful and unsuccessful runs it occurs, and the total number of successful and unsuccessful runs. The elements of the cleaned message list are the result of the log parser that is used to determine the log's log type.

Here is an example of a run with one log in a YAML format:

```
9e6a84eb-1d54-4ded-b362-a351edf86b48:
  succeeded: true
    logs:
        33f46e06-cb9d-4efa-be7c-4265c7dd5ec0:
          timestamp: 2022-06-09 15:26:59.595764+00:00
          message: 'Annotations:  acquireTime:
                       2022-06-09T15:26:59.595764+00:00
                   leader: application-manager-postgres-0
                   optime: 51950056'
          cleaned_message:
          - annotations
          - acquiretime
          - __timestamp__
          - leader
          - application-manager-postgres-__num__
          - optime
          - __num__
          log_type:
            uuid: ee31cdc6-fbf9-4c8f-be34-c66aacaeb8cd
```

```
n_successful_runs: 51
n_failed_runs: 4
n_total_successful_runs: 51
n_total_failed_runs: 10
```

The runs and log types are distributed in the following way:

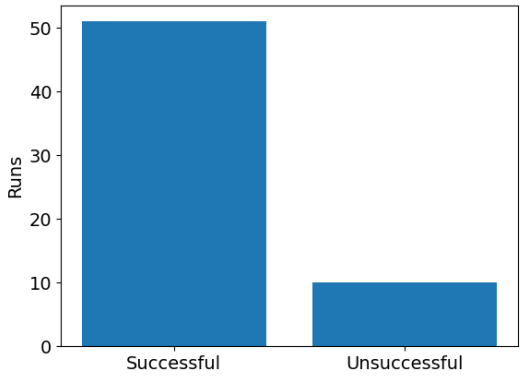

Figure 3: Distribution of runs

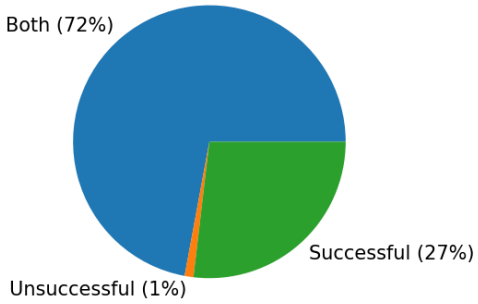

Figure 4: Distribution of log types over runs

As shown above there are a lot more successful runs than there are unsuccessful runs. The 61 runs are divided into 51 successful runs and 10 unsuccessful runs. This also reflects in the distribution of the log types. There are 2859 different log types in total. As there are more successful runs than there are unsuccessful runs and unsuccessful runs usually end sooner than successful runs, the amount of log types that occur only in successful runs is a lot higher than the ones that occur in unsuccessful runs. Nevertheless, most of the log types occur in both types of runs. As most of the logs occur before a part of the pipeline fails and when a part of the pipeline fails, it usually creates only a few logs that indicate the failure, this was the expected outcome.

The following diagram shows the distribution of the number of runs that log types occur in:

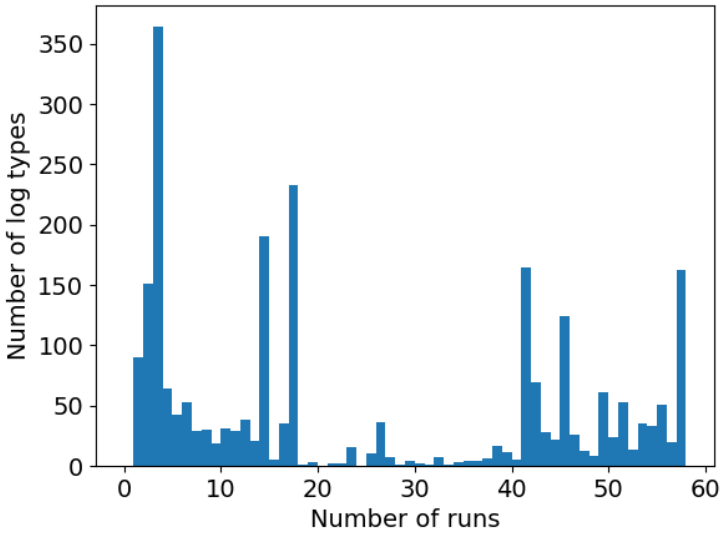

Figure 5: Occurrences of log types in runs

As seen in figure 5 the log type's distribution over the runs is quite the opposite of a normal distribution. There are many log types occurring either in a lot of runs or only in a few runs, while log types that occur in half of the runs are pretty rare. This distribution can be explained in the following way. On the one hand, we have very specific logs that only occur in a few runs, and on the other hand, we have very common logs that occur nearly every time.

Log types that only occur in a few runs do not occur in a regular manner. That is why I filtered them out of the dataset for the following analysis regarding means and standard deviations.

3.2 Analysis of Dataset

First I want to take a look into the frequency that log types occur in runs. Because every log type occurs in multiple runs, I calculated its mean frequency over the runs and its standard deviation. The results are plotted in boxplots:

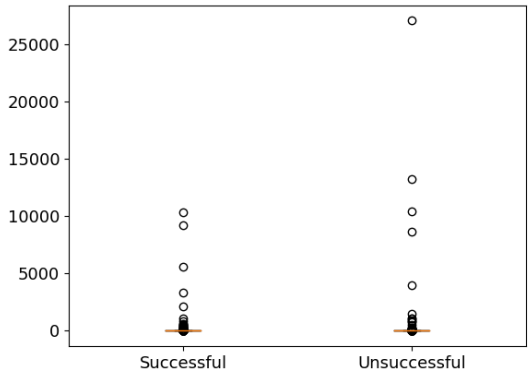

Figure 6: means of log type frequencies

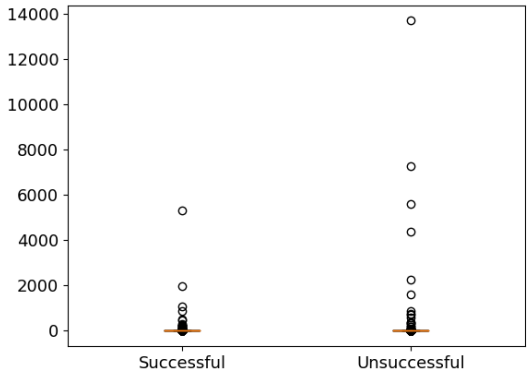

Figure 7: standard deviations of log type frequencies

As seen in figure 6, there are some very high outliers. The outliers of unsuccessful runs are much higher, which could indicate the source of an error. By taking a look at the messages of the log types with the two highest means we get:

- ”config-map with name : ’eric-oss-common-base-evnfm-nbi-config-kubernetes’ not present in namespace : ’evnfm-staging”’ [...] ”severity”:”info”
- WARN [cfgwarn] docker/input.go:49 DEPRECATED: ’docker’ input deprecated. Use ’container’ input instead. Will be removed in version: 8.0.0

Both messages are not very detrimental for a run. The first one is classified as info which already indicates that it probably will not have a huge effect on the run. The second one is classified as warning which is alarming at first look. But taking a closer

look at the message shows that it is a version deprecation warning. This is a very common warning for most developers and has no effect on the outcome of the run. The fact that these two log types are also the most frequent ones in successful runs supports the theory that these log types are just very common and not detrimental.

The standard deviation's outliers in figure 7 have got a very similar distribution. Taking a look at their messages shows that these are the same ones as the mean's outliers in figure 6. This shows that there are no detrimental log types here either.

Because the outliers do not have a big impact on the run, they can be removed from the boxplot.

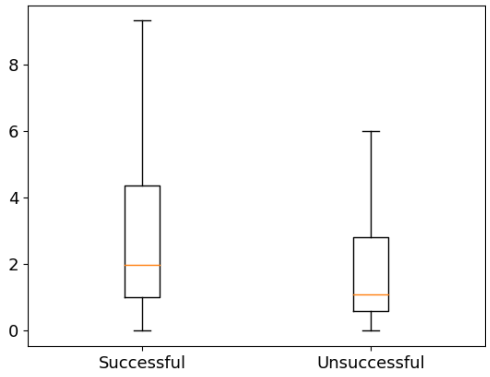

Figure 8: means of log types frequencies filtered outliers

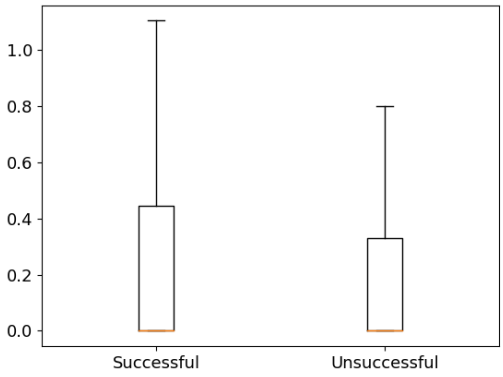

Figure 9: standard deviations of log types frequencies filtered outliers

Figure 8 shows the same data as figure 6, but does not show its outliers. In contrast to the outliers seen in figure 6 the means of log types in successful are higher than in unsuccessful runs. This can be explained by unsuccessful runs ending sooner than successful runs.

It also can be seen that most log types have means that are much lower than they

seemed in figure 6. For the log type's occurrences in successful runs, more than half of the log types have a mean equal to or smaller than two. As the lower quartile is at one, at least 25 percent of the log types have a mean between one and two. The means for the log type's occurrence in unsuccessful runs are even lower. Here 75 percent of the log types have means between zero and three, while the lower 50 percent are distributed below one.

Figure 9 shows the standard deviations without the outliers. Here the log types have in both, successful and unsuccessful, runs a very small standard deviation. That means the log types frequency does not vary much over the runs.

Summing up the information from the means and standard deviations, log types occur mostly one or two times in runs which does not vary much in different runs.

To further analyze the log type's frequency, I've compared their mean frequency and the number of runs it occurs in for successful and unsuccessful runs.

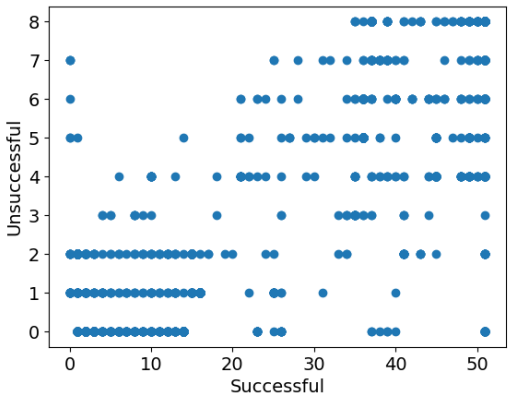

Figure 10: Distribution of log type in successful and unsuccessful runs

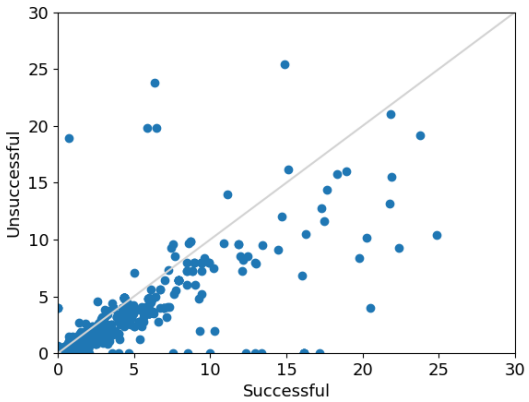

Figure 11: comparison of means zoomed

Figure 10 compares in how many successful or unsuccessful runs each log type occurs.

For example, the dot in the upper left corner means that one log type occurs in seven unsuccessful runs but in no successful runs.
What sticks out most is that the density is highest in the lower left and upper right corners. This again shows that they are either very common and occur in almost every run or very uncommon and occur in almost no run regardless of the run's outcome.
The density is lowest in the other two corners. Log types that fall into one of these corners either have a high occurrence in successful and a low occurrence in unsuccessful runs or the other way around. These log types are very interesting as they might indicate the outcome of the run.

Figure 11 shows a comparison of the average occurrence of log types in successful and unsuccessful runs. As figure 8 shows that most points fall into the area between 0 to 10, the diagram is cut off after 30 to establish more visibility but also include some of the minor outliers.
Because most log types occur pretty much the same amount in failed and successful runs, most of the points are centered around the bisector. Nevertheless, there is a slight tendency to the successful axis because successful runs usually run a bit longer.
The more interesting log types are the ones that are lying on the axes. These are the ones that occur in either successful or in failed runs, but not in both. For those lying on the unsuccessful axis, these log types could have information, on why their runs failed, as they occur only in unsuccessful runs. Those who are lying on the successful axis could be logs that are missing in the failed runs, which could have led to the failure of the run.

There are six log types lying on the unsuccessful axis. These are also the ones that

are in the upper left corner in figure 10. Three of them are related to one event about successfully creating a service account token. These logs do not indicate the cause of an error. By taking a look into the original dataset, I saw that the logs that occurred right before this event were in most cases about an error. That means that this event is most likely a part of the clean-up process that takes place right after a crash happens to ensure there is no unwanted temporary data remains in the environment.
The next log's cleaned message is: '__timestamp__ [INFO] expiration: revoked lease: lease_id=__path__'. This message indeed can indicate an error despite being classified as info. Its surrounding logs are regular info logs. That brings me to the conclusion that there is no serious effect coming from the log that can lead to the failure of the runs it occurs in.
The two remaining log messages do not give too much information except being a warning. That means it could be related to an error but it is pretty unlikely, as they do not give any information.

Taking a look at the log types lying on the successful axis can bring information about logs that belong to missing parts of the pipeline for the run to be successful. There are 59 logs that fall into this category.
One of the log type's message is: '[192.168.210.149]:5701 [dev] [3.12.5] Initializing cluster partition table arrangement...","logger":"com.hazelcast.internal.partition.impl. PartitionStateManager'. Its severity is classified as info. As it occurs in every run one time, it is likely to be important to have it. It can indicate failure of the run if it is missing because the process behind the log type probably is important and was not executed. This would be very important information for the developer when it comes to debug-

ging.

Another interesting finding within these log types are logs indicating health checks. Health checks send requests to a backend with specific parameters. Based on the backend's response, the health status is calculated then to determine the availability for each backend.[Goo] There are eight log types about requests for health checks that occur in no unsuccessful run but in each successful run. Their mean frequency reaches from 50 up to 575 occurrences per run. That means that all unsuccessful runs miss some health checks, that are done regularly in successful runs.

4 Conclusion

This thesis analyzed frequencies of log types to provide background information if considering them in the abnormality rating could help to improve it.

Therefore I have analyzed a dataset with several methods of statistics. I first calculated the distribution of logs over the runs to establish an overview. The distribution showed that log types either show up very often or very rarely. Then I calculated for each log type that occurs in more than half of the runs its average frequency in the runs. Most log types occur on average only one or two times per run. There are some very big outliers that occur over 10,000 times. Nevertheless, they could not be connected with an error, as these were very common log messages. To analyze the remaining log types, I have compared their mean frequency in successful runs with their mean frequency in unsuccessful runs, by plotting both in a scatter plot. There the log types can be clustered into three groups: one includes the log types that lay at the bisector and the other two groups include the ones that lay on each axis. The important ones are the ones lying in the axes, as these occur either in successful or unsuccessful runs, but not in both.

The log types that do not occur in successful runs have higher chances to be related to an error. But this information alone is not enough to classify it as abnormal. Additionally looking for keywords, like error or failed, could filter the log types for abnormalities. The log types that occur in each successful run, but in no unsuccessful runs, are likely to indicate processes that were not executed correctly. The impact of these missing log types needs to be further analyzed because many log types that occur very late in the runs do not occur in unsuccessful runs, but also do not contribute to their failure.

In conclusion, taking the frequencies of log types into account can help to improve the abnormality rating. But with an abnormal frequency of a log type, the log type's abnormality cannot be determined on its own because not all log types with an abnormal frequency are abnormal log types. The reason for an abnormal frequency often is another process laying behind it that ran differently than expected.
Taking log types that occur in only one type of run brings abnormal log types that are likely to play an important role in the failure of a run but also log types that are completely normal. To find the abnormal ones among them Lexicon could search for keywords like failed or error for the ones only occurring in unsuccessful runs as these likely indicate a failed process. For the ones that only occur in successful runs, it could be searched for keywords like completed or health because these log types could be important and could be missing in unsuccessful runs.
This can help to find logs that are completely missing and also makes it easier to find abnormal logs that did not occur in successful runs but unsuccessful runs.

In later work, the effect of frequencies differing from the mean frequency of single log types can be examined. To do so, a dataset with many more unsuccessful runs is needed to have enough logs with abnormal frequencies. Doing that could show which logs have a detrimental effect if occurring some more or fewer times in a run.

References

[Bah22] Tanuja Bahirat. *Introduction to Virtualization in Cloud Computing types and benefits*. Last visited: 23.11.2022. 2022. URL: https://www.mygreatlearning.com/blog/virtualization-in-cloud-computing/#:~:text=Concerning%5C20Cloud%5C%20Computing%5C%2C%5C%20virtualization%5C%20is,users%5C%20respectively%5C%20using%5C%20their%5C%20machines..

[Cun09] Michael Zhivich; Robert K. Cunningham. "The Real Cost of Software Errors". In: (2009). DOI: 10.1109/MSP.2009.56.

[Ebe14] Collin Eberhardt. *The Art of Logging*. Last visited: 22.11.2022. 2014. URL: https://www.codeproject.com/Articles/42354/The-Art-of-Logging#what.

[Gee21] GeeksForGeeks. *Logging in Python*. Last visited: 22.11.2022. 2021. URL: https://www.geeksforgeeks.org/logging-in-python/#:~:text=Logging%5C%20is%5C%20a%5C%20means%5C%20of,the%5C%20cause%5C%20of%5C%20the%5C%20problem..

[Gee22] GeeksForGeeks. *Virtualization In Cloud Computing and Types*. Last visited: 23.11.2022. 2022. URL: https://www.geeksforgeeks.org/virtualization-cloud-computing-types/.

[Goo] Google. *Health checks overview*. Last visited: 13.12.2022. URL: https://cloud.google.com/load-balancing/docs/health-check-concepts.

[Hay22] Adam Hayes. *Descriptive Statistics: Definition, Overview, Types, Example*. Last visited: 22.11.2022. 2022. URL: https://www.investopedia.com/terms/d/descriptive_statistics.asp.

[MG11] "Peter Mell and Timothy Grance". ""The NIST Definition of Cloud Computing"". In: (2011). DOI: 10.6028/NIST.SP.800-145.

[Pen21] Fiorella Zampetti; Salvatore Geremia; Gabriele Bavota; Massimiliano Di Penta. "CI/CD Pipelines Evolution and Restructuring: A Qualitative and Quantitative Study". In: (2021). DOI: 10.1109/ICSME52107.2021.00048.

[Sar21] Ajay Sarangam. *Types Of Cloud Environment – A Simple Overview*. Last visited: 22.11.2022. 2021. URL: https://www.jigsawacademy.com/blogs/cloud-computing/cloud-environment/.

[Stu] StudySmarter. *Charts and Diagrams*. Last visited: 03.12.2022. URL: https://www.studysmarter.co.uk/explanations/math/probability-and-statistics/charts-and-diagrams/.

[Yel18] Parampreet Kaur; Jill Stoltzfus; Vikas Yellapu. "Descriptive statistics". In: (2018). URL: https://www.ijam-web.org/article.asp?issn=2455-,5568;year=2018;volume=4;issue=1;spage=60;epage=63;aulast=Kaur.

[YPZ12] Ding Yuan, Soyeon Park, and Yuanyuan Zhou. "Characterizing logging practices in open-source software". In: (2012). DOI: 10.1109/ICSE.2012.6227202.

[Zha15] Jieming Zhu; Pinjia He; Qiang Fu; Hongyu Zhang; Michael R. Lyu; Dongmei Zhang. "Learning to Log: Helping Developers Make Informed Logging Decisions". In: (2015). DOI: 10.1109/ICSE.2015.60.