Master's Degree in Informatics Engineering

Final Dissertation

Observing and Controlling Performance in Microservices

Author:

André Pascoal Bento

Supervisor:

Prof. Filipe João Boavida Mendonça Machado Araújo

Co-Supervisor:

Prof. António Jorge Silva Cardoso

FACULDADE DE CIÊNCIAS E TECNOLOGIA

UNIVERSIDADE DE COIMBRA

July 2019

This page is intentionally left blank.

Abstract

Microservice based software architecture are growing in usage and one type of data generated to keep history of the work performed by this kind of systems is called tracing data. Tracing can be used to help Development and Operations (DevOps) perceive problems such as latency and request work-flow in their systems. Diving into this data is difficult due to its complexity, plethora of information and lack of tools. Hence, it gets hard for DevOps to analyse the system behaviour in order to find faulty services using tracing data. The most common and general tools existing nowadays for this kind of data, are aiming only for a more human-readable data visualisation to relieve the effort of the DevOps when searching for issues in their systems, however these tools do not provide good ways to filter this kind of data neither perform any kind of tracing data analysis and therefore, they do not automate the task of searching for any issue presented in the system, which stands for a big problem because they rely in the system administrators to do it manually. In this thesis is present a possible solution for this problem, capable of use tracing data to extract metrics of the services dependency graph, namely the number of incoming and outgoing calls in each service and their corresponding average response time, with the purpose of detecting any faulty service presented in the system and identifying them in a specific time-frame. Also, a possible solution for quality tracing analysis is covered checking for quality of tracing structure against OpenTracing specification and checking time coverage of tracing for specific services. Regarding the approach to solve the presented problem, we have relied in the implementation of some prototype tools to process tracing data and performed experiments using the metrics extracted from tracing data provided by Huawei. With this proposed solution, we expect that solutions for tracing data analysis start to appear and be integrated in tools that exist nowadays for distributed tracing systems.

Keywords

Microservices, Cloud Computing, Observability, Monitoring, Tracing.

This page is intentionally left blank.

Resumo

A arquitetura de software baseada em micro-serviços está a crescer em uso e um dos tipos de dados gerados para manter o histórico do trabalho executado por este tipo de sistemas é denominado de tracing. Mergulhar nestes dados é díficil devido à sua complexidade, abundância e falta de ferramentas. Consequentemente, é díficil para os DevOps de analisarem o comportamento dos sistemas e encontrar serviços defeituosos usando tracing. Hoje em dia, as ferramentas mais gerais e comuns que existem para processar este tipo de dados, visam apenas apresentar a informação de uma forma mais clara, aliviando assim o esforço dos DevOps ao pesquisar por problemas existentes nos sistemas, no entanto estas ferramentas não fornecem bons filtros para este tipo de dados, nem formas de executar análises dos dados e, assim sendo, não automatizam o processo de procura por problemas presentes no sistema, o que gera um grande problema porque recaem nos utilizadores para o fazer manualmente. Nesta tese é apresentada uma possivel solução para este problema, capaz de utilizar dados de tracing para extrair metricas do grafo de dependências dos serviços, nomeadamente o número de chamadas de entrada e saída em cada serviço e os tempos de resposta coorepondentes, com o propósito de detectar qualquer serviço defeituoso presente no sistema e identificar as falhas em espaços temporais especificos. Além disto, é apresentada também uma possivel solução para uma análise da qualidade do tracing com foco em verificar a qualidade da estrutura do tracing face à especificação do OpenTracing e a cobertura do tracing a nível temporal para serviços especificos. A abordagem que seguimos para resolver o problema apresentado foi implementar ferramentas protótipo para processar dados de tracing de modo a executar experiências com as métricas extraidas do tracing fornecido pela Huawei. Com esta proposta de solução, esperamos que soluções para processar e analisar tracing comecem a surgir e a serem integradas em ferramentas de sistemas distribuidos.

Palavras-Chave

Micro-serviços, Computação na nuvem, Observabilidade, Monitorização, Tracing.

This page is intentionally left blank.

Acknowledgements

This work would not be possible to be accomplished without effort, help and support from my family, fellows and colleagues. Thus, in this section I would like to give my sincere thanks to all of them.

Starting by giving thanks to my mother and to my whole family, who have supported me through this entire and long journey, and who always gave and will always give me some of the most important and beautiful things in life, love and friendship.

In second place, I would like to thank all people that were involved directly in this project. To my supervisor, Professor Filipe Araújo, who contributed with his vast wisdom and experience, to my co-supervisor, Professor Jorge Cardoso, who contributed with is vision and guidance about the main road we should take and to Engineer Jaime Correia, who "breathes" these kind of topics through him and helped a lot with is enormous knowledge and enthusiasm.

In third place, I would like to thank Department of Informatics Engineering and the Centre for Informatics and Systems, both from the University of Coimbra, for allowing and provide the resources and facilities to to be carried out this project.

In fourth place, to the Foundation for Science and Technology (FCT), for financing this project facilitating its accomplishment, to Huawei, for providing tracing data, core for this whole research, and to Portugal National Distributed Computing Infrastructure (INCD) for providing hardware to run experiments.

And finally, my sincere thanks to everyone that I have not mentioned and contributed to everything that I am today.

This page is intentionally left blank.

Contents

This page is intentionally left blank.

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

This document presents the *Master Thesis* in *Informatics Engineering* of the student *André Pascoal Bento* during the school year of 2018/2019, taking place in the *Department of Informatics Engineering (DEI)* of the *University of Coimbra*.

1.1 Context

In today's world, software systems tend to become more distributed as time move on, resulting in new approaches that lead to new solutions and new patterns of developing software. One way to solve this is to develop systems that have their components decoupled, creating software with "small pieces" connected to each other that encapsulate and provide a specific function in the larger service. This way of developing software is called Microservices and has become mainstream in the enterprise software development industry [1]. However, with this kind of approach, the systems complexity is increased as a whole because with more "small pieces", more connections are needed and with this more problems related to latency and requests become harder to detect, analyse and correct [2].

To keep a history of the work performed by this kind of systems, multiple techniques like monitoring [3], logging [4] and tracing [5] are adopted. Monitoring consists on measuring some aspects like, e.g., Central Processing Unit (CPU) usage, hard drive usage and network latency of the entire system or of some specific node in a distributed system. Logging provides an overview to a discrete, event-triggered log. Finally, tracing is much similar to logging, however the focus is registering the flow of execution of the program through several system modules and boundaries. Lastly, distributed tracing, shares the focus on preserving causality relationships, however, is geared towards the modern distributed environments, where state is partitioned over multiple, threads, processes, machines and even geographical locations. This last one is better explained in Subsection 3.1.3 - Distributed Tracing. There are multiple approaches to gather information of this kind of systems, each with its benefits and disadvantages.

The main problem with this nowadays is that there are not many implemented tools for processing tracing data and none for performing analysis of this type of data. For monitoring it tend to be easier, because data is represented in charts and diagrams, however for logging and tracing it gets harder to manually analyse the data due to multiple factors like its complexity, plethora and increasing quantity of information. There are some visualisation tools for the Development and Operations (DevOps) to use, like the ones presented in Subsection 3.2.1 - Distributed Tracing Tools, however none of them gets to the point of performing the analysis of the system using tracing, has they tend to be developed only for visualisation and display of tracing data in a more human readable way. Nevertheless, this is critical information about the system behaviour, and thus there is the need for performing automatic tracing analysis.

1.2 Motivation

The motivation behind this work resides in exploring and develop ways to perform tracing analysis in microservice based systems. The analysis of this kind of systems tend to be very complex and hard to perform due to their properties and characteristics, as it is explained in Subsection 3.1.1 - Microservices, and to the type of data to be analysed, presented in Subsection 3.1.3 - Distributed Tracing.

DevOps teams have lots of problems when they need to identify and understand problems with this systems. They usually detect the problems when the client complains about the quality of service, and after that DevOps dive in monitoring metrics like, e.g, CPU usage, usage, hard drive usage and network latency, and then in distributed tracing data visualisations and logs to find some explanation to what is causing the reported problem. This involves a very hard and tedious work of look-up through lots of data that represents the history of work performed by the system and, in most cases, this tedious work reveals like a big "find a needle in the haystack" problem. Some times, DevOps can only perceive problems in some services and end up "killing" and rebooting these services which is wrong, however, due to lack of time and difficulty in identifying anomalous services precisely this is the best known approach.

Problems regarding the system operation are more common in distributed systems and their identification must be simplified. This need of simplification comes from the exponential increase in the amount of data needed to retain information and the increasing difficulty in manually managing distributed infrastructures. The work presented in this thesis, aims to perform a research around these needs and focus on presenting some solutions and methods to perform tracing analysis.

1.3 Goals

The main goals for this thesis consists on the main points exposed bellow:

1. Search for existing technology and methodologies used to help DevOps teams in their current daily work, with the objective of gathering the best practices about handling tracing data. Also, we aim to understand how these systems are used, what are their advantages and disadvantages to better know how we can use them to design and produce a possible solution capable of performing tracing analysis. From this we expect to learn the state of the field for this research, covering the core concepts related work and technologies, presented in Chapter 3 - State of the Art.

2. Perform a research about the main needs of DevOps teams, to better understand what are their biggest concerns that lead to their approaches when performing pinpointing of microservices based systems problems. Relate these approaches with related work in the area, with the objective of understanding what other companies and groups have done in the field of automatic tracing analysis. The processes used to tackle this type of data, their main difficulties and conclusions provide a

better insight about the problem. From this we expected to have our research objectives clearly defined and a compilation of questions to be evaluated and answered, presented in Chapter 4 - Research Objectives and Approach.

3. Reason about all the gathered information, design and produce a possible solution that provides a different approach to perform tracing analysis. From this we expect first to propose a possible solution, presented in Chapter **??**. The we implement it using state of the art technologies, feed it with tracing data provided by Huawei and collect results, presented in Chapter **??** - **??**. Finally, we provide conclusions to this research work in the last Chapter **??** - **??**.

1.4 Research Contributions

From the work presented on this thesis, the following research contributions were made:

- Andre Bento, Jaime Correia, Ricardo Filipe, Filipe Araujo and Jorge Cardoso. On the Limits of Automated Analysis of OpenTracing. International Symposium on Network Computing and Applications (IEEE NCA 2019) (The paper is waiting review).

1.5 Document Structure

This section presents the document structure in this report, with a brief explanation of the contents in every section. The current document contains a total of seven chapters, including this one, Chapter 1 - Introduction. The remaining six of them are presented as follows:

- In Chapter 2 - Methodology are presented the elements involved in this work, with their contributions, has well as the work plan, with “foreseen” and “real” work plans comparison and analysis.
- In Chapter 3 - State of the Art the current state of the field for this kind of problem is presented. This chapter is divided in three sections. The first one, Section 3.1 - Concepts introduces the reader to the core concepts to know as a requirement for a full understanding of the topics discussed in this thesis. The second, Section 3.2 - Technologies presents the result of a research for current technologies, that are able to help solving this problem and produce a proposed solution to be implemented. Finally, Section 3.3 - Related Work presents the reader to related researches produced in the field of distributed tracing data handling.
- In Chapter 4 - Research Objectives and Approach we present how we tackled this problem, the main difficulties that were found and the objectives involved to solve the issues that are presented. Also, in this chapter, a compilation of questions are presented and evaluated with some reasoning about possible ways to answer them.
- In Chapter **??** - **??** a possible solution for the presented problem is exposed and explained in detail. This chapter is divided in four sections. The first one, Section **??** - **??**, expose the functional requirements with their corresponding priority levels and a brief explanation to every single one of them. The second one, Section **??** - **??**, contains the gathered non-functional requirements that were used to

build the solution architecture. The third one, Section ?? - ??, presents the defined technical restrictions for this project. The last one, Section ?? - ??, presents the possible solution architecture using some representational diagrams, and ends with an analysis and validation to check if the presented architecture meets up the restrictions involved in the architectural drivers.

- In Chapter ?? - ??, the implementation process of the possible solution is presented with detail. This chapter is divided in three main sections covering the whole implementation process, from the input data set through the pair of components presented in the previous chapter. The first one, Section ?? - ??, the tracing data set provided by Huawei to be used as the core data for research is exposed with some detail. Second, in Section ?? - ?? we present the possible solution for the first component, namely "Graphy OpenTracing processor (OTP)", that processes and extracts metrics from tracing data. The final Section ?? - ?? presents the possible solution for the second component, namely "Data Analyser", that handles data produced by the first component and produces the analysis reports. Also, in the last two sections presented, the used algorithms and methods in the implementations are properly detailed and explained.

- In Chapter ?? - Research Objectives and Approach, the gathered results, corresponding analysis and limitations of tracing data are presented. This chapter is divided in three main sections. The first one, Section ?? - ??, the results regarding the gathered observations on the extracted metrics of anomalous service detection are presented and explained. Second, in Section ?? - ?? the results obtained from the quality analysis methods applied to the tracing data set are presented and explained. The final Section ?? - ?? we present the limitations felted when designing a solution to process tracing data, more precisely OpenTracing data.

- Last, in Chapter ?? - ??, the main conclusions for this research work are presented. The chapter is divided in three main sections. First, Section ?? - ??, a reflection about the implemented tools, methods produced and the open paths from this research are exposed. Also a reflection of the main difficulties felted with this research regarding the handling of tracing data are presented. Second, Section ?? - ??, the future work that can be addressed considering this work is properly explained taking into consideration what is said in the previous section. Finally, Section ?? - ??, the state of answers for the selected questions defined in this research are discussed.

Next, Chapter 2 - Methodology, the elements involved in this work, their contributions and work plans for this research project are presented.

Chapter 2

Methodology

The methodology of work carried out in this research project is presented in this chapter. First, every member involved will be mentioned as well as their individual contribution for the project. Second, the adopted approach and organisation process of the collaborators involved will be explained. Finally, the work plan as well as the work performed, including the foreseen and real work plans for the whole year of work are presented and discussed.

The main people involved in this project were myself, André Pascoal Bento, student at the Master course of Informatics Engineering at Department of Informatics Engineering (DEI), who carried out the investigation and development of the project. In second, Prof. Filipe Araújo, assistant professor at the University of Coimbra, who contributed with his vast knowledge and guidance on topics about distributed systems and cloud computing. In third, Prof. Jorge Cardoso, Chief Architect for Intelligent CloudOps at Huawei Technologies, who contributed with his vision, great contact with the topics addressed in this work and with the tracing data set from Huawei Cloud Platform [6]. In fourth, Eng. Jaime Correia, doctoral student at DEI, who contributed with his vast technical knowledge regarding the topics of tracing and monitoring microservices. In Sixth, Eng. Ricardo Filipe, doctoral student at DEI, who contributed, like the ones mentioned before, with peer review of the produced paper for the International Symposium on Network Computing and Applications (IEEE NCA 2019).

This work stands for an investigation and was mainly an exploratory work, therefore, no development methodology was adopted. Meetings were scheduled to happen every two weeks. In these meetings, every participant element in the project joined with the objective of discussing the work carried out in the last two weeks and define the new course of research. In the first semester, topics like published papers, state of the art, analysis of related work and a proposition of solution were the main focus. In the second semester, two more colleagues joined the whole project (DataScience4NP). One of them with a project somehow related to this research. They started participating in meetings and this contributed with a wider discussion of ideas. In these meeting, the main topics covered were: implementation of the proposed solution, research for algorithms and methods for trace processing and analysis of gathered data. In the end, these meetings were more than enough to keep the productivity and good work.

Total time spent in each semester, by week, were sixteen (16) hours for the first semester and forty (40) hours for the second one. In the end, it was spent a total of three-hundred and four (304) hours for the first semester, starting in 11.09.2018 and ending in 21.01.2019 (19 weeks $*$ 16 hours per week). For the second semester, eight-hundred and forty (840) hours were spent, starting in 04.02.2019 and ending in 28.06.2019 (21 weeks $*$ 40 hours per week).

Before starting this research project, there was a work plan defined for two semesters presented in the proposition. For record, these plans are presented in Figure 2.1.

For purposes of analysis and comparison with the proposed work plan, the real work plan carried out in the first semester is presented in Figure 2.2.

As we can see in Figures 2.1 and 2.2, the proposed work for the first semester has suffered some changes, when comparing it to the real work plan. Task 1 - Study the state of the art(...), was branched in two, 1 - Project Contextualisation and Background and 2 - State of the Art, however, these last ones tocked more time to accomplish due to lack of work in the field of trace processing and trace analysis, core topics for this thesis. Task 2 - Integrate the existing work was replaced by task 3 - Prototyping and Technologies Hands-On due to redirections in the work course. This redirection was done due to interest increase in testing state of the art technologies, allowing us to get a better visualisation of the data provided by Huawei and enhancing our investigation work. The remaining tasks took almost the predicted time to accomplish.

For the second semester, an “expected” work plan was defined with respect to the proposed work, presented in Figure 2.1, and the state of the research at the time. The expected work plan can be visualized in Figure 2.3. This Figure contains the expected (Grey) and real (Blue) work for the second semester.

Three main changes were made over time in the work plan. The first one involved a reduction in task 1 - Metrics collector tool. When the solution was being implemented and the prototype was capable to extract a set of metrics, we decided to stop the implementation process to analyse the research questions. Second, this analysis lead to an emergence of ideas, “2 - Restructuring research questions’,’ and thus a project redirection. Tests were removed from planning and the project followed with the objective of producing the data analyser, “3 - Data Analyser tool”, and with it, answer two main questions regarding anomalous services and quality of tracing. Third, the introduction of a new task, “4 - Write paper to NCA 2019”, covering the work presented in this thesis.

These Figures have been created by an open-source tool called GanttProject [7] that produce Gantt charts, a kind of diagram used to illustrate the progress of the different stages of a project.

Next, Chapter 3 - State of the Art, the state of the field is covered with core concepts, technologies and related work.

Name	Begin date	End date
1 - Study the state of the art	9/11/18	11/5/18
2 - Integrate the existing work	11/6/18	12/3/18
3 - Define requirements of the monitoring tool	12/4/18	12/31/18
4 - Write intermediate report	1/1/19	1/21/19
5 - Implement the monitoring tool	2/4/19	3/29/19
6 - Test and evaluate the results	4/1/19	5/24/19
7 - Write final report	5/27/19	6/28/19

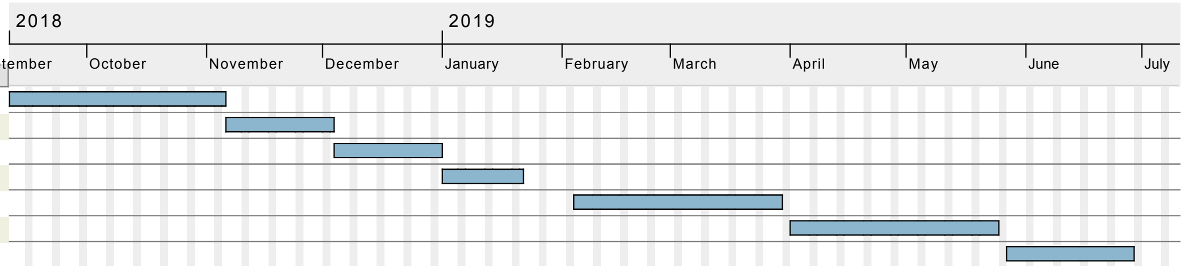

Figure 2.1: Proposed work plan for first and second semesters.

Name	Begin date	End date
1 - Project Contextualization and Background	9/11/18	9/24/18
2 - State of the Art	9/25/18	11/19/18
2.1 - Concepts	9/25/18	10/15/18
2.2 - Technologies	10/16/18	11/19/18
3 - Prototyping and Technologies Hands-On	11/20/18	12/3/18
4 - Solution Specification	12/4/18	12/24/18
4.1 - Gathering Requirements	12/4/18	12/17/18
4.2 - Building Architecture	12/18/18	12/24/18
5 - Writing of Intermediary Report	12/25/18	1/21/19

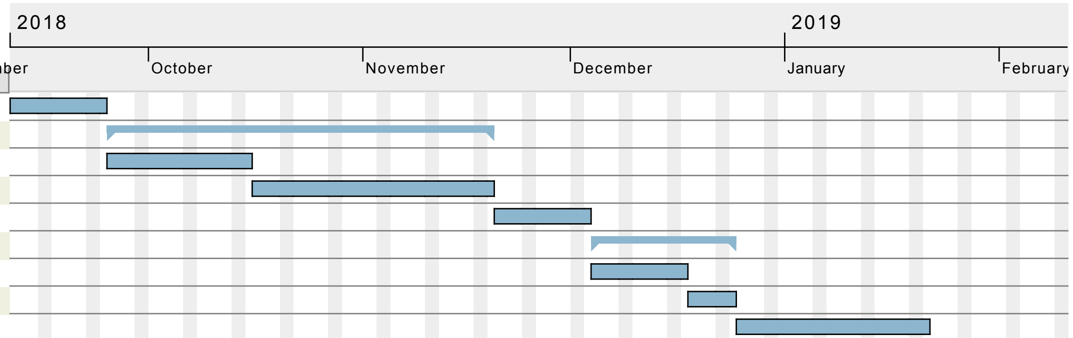

Figure 2.2: Real work plan for first semester.

Name	Begin date	End date
1.a - Metrics collector tool (Expected)	2/4/19	5/3/19
1.1 - Setup project (Expected)	2/4/19	2/6/19
1.2 - Implement Controller (Expected)	2/7/19	2/18/19
1.3 - Implement Comunication (Expected)	2/19/19	2/22/19
1.4 - Implement File IO (Expected)	2/25/19	2/27/19
1.5 - Setup databases (Expected)	2/28/19	3/4/19
1.6 - Implement database repositories (Expected)	3/5/19	3/11/19
1.7 - Implement Graph Processor (Expected)	3/12/19	3/21/19
1.8 - Implement Logging Component (Expected)	3/22/19	3/26/19
1.9 - Research apropriate analysis algorithms (Expected)	3/27/19	4/11/19
1.10 - Implement Data Analyser (Expected)	4/12/19	4/23/19
1.11 - Define tests to be performed (Expected)	4/24/19	4/26/19
1.12 - Implement Testing Component (Expected)	4/29/19	5/3/19
1.b - Metrics collector tool	2/4/19	3/15/19
1.1 - Setup project (Real)	2/4/19	2/6/19
1.2 - Implement Logging Component (Real)	2/7/19	2/7/19
1.3 - Setup Docker Containers (Real)	2/8/19	2/8/19
1.4 - Implement Controller (Real)	2/11/19	2/20/19
1.5 - Implement File IO (Real)	2/21/19	2/25/19
1.6 - Implement Processors (Real)	2/26/19	3/4/19
1.7 - Setup databases (Real)	3/5/19	3/7/19
1.8 - Implement database repositories (Real)	3/8/19	3/12/19
1.9 - Implement metrics storage (Real)	3/13/19	3/15/19
2.a - Test and evaluate results (Expected)	5/6/19	5/27/19
2.1 - Run tests (Expected)	5/6/19	5/15/19
2.2 - Write analysis results (Expected)	5/16/19	5/27/19
2.b - Restructure research questions (Real)	3/18/19	3/27/19
2.1 - Write question analysis report (Real)	3/18/19	3/22/19
2.2 - Report review (Real)	3/25/19	3/27/19
3.a - Write final report (Expected)	5/28/19	6/28/19
3.b - Data analysis tool (Real)	3/28/19	5/17/19
3.1 - Quality of tracing analysis (Real)	3/28/19	4/3/19
3.1.1 - Time coverability testing (Real)	3/28/19	4/1/19
3.1.2 - Structure testing (Real)	4/2/19	4/3/19
3.2 - Research appropriate analysis algorithms (Real)	4/4/19	4/19/19
3.3 - Setup Jupyter notebooks (Real)	4/22/19	4/24/19
3.4 - Implement proposed solution and gather results (Real)	4/25/19	5/17/19
3.4.1 - Is there any anomalous service? (Real)	4/25/19	5/10/19
3.4.2 - Results gathering (Real)	5/13/19	5/17/19
4 - Write paper for NCA 2019 (Real)	5/20/19	5/29/19
5 - Write final report (Real)	5/30/19	6/28/19

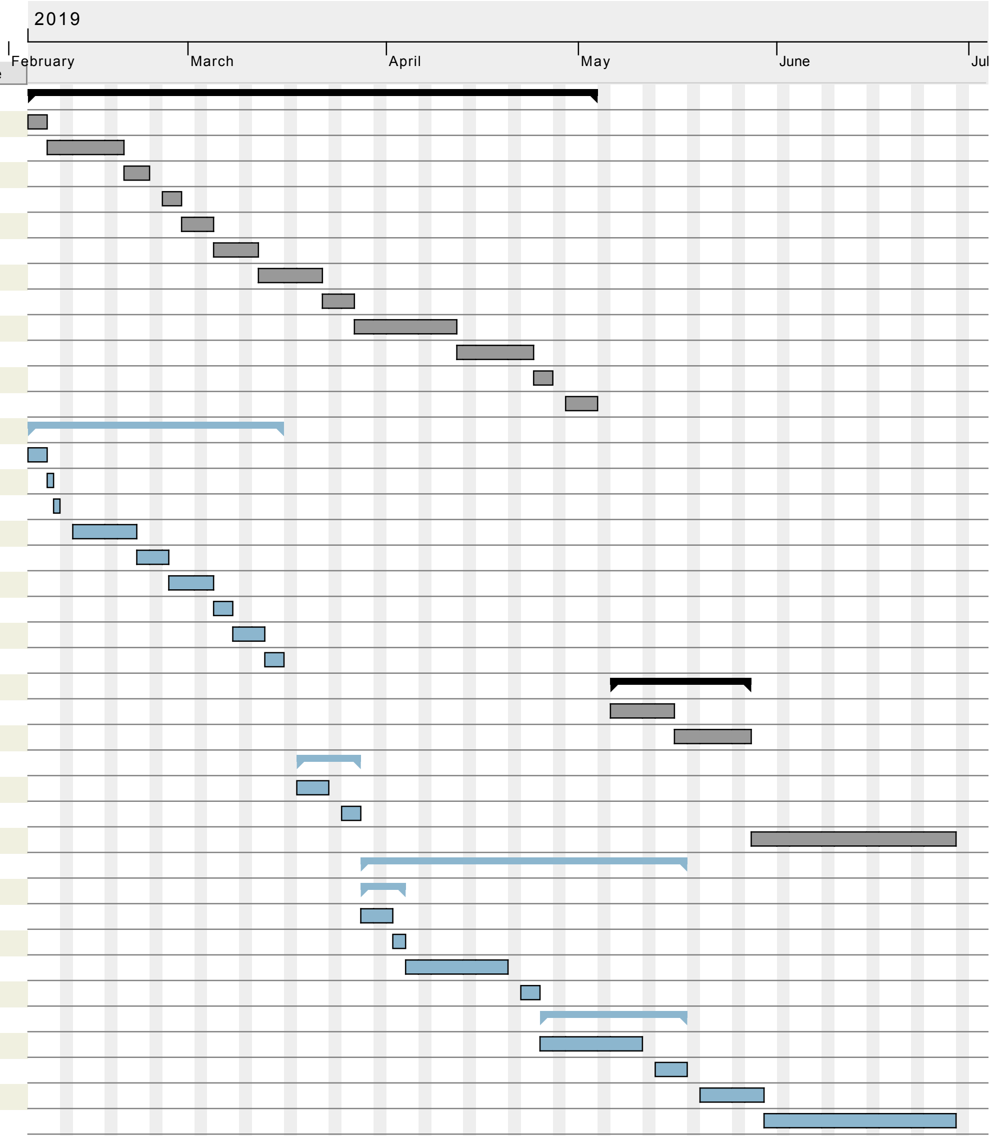

Figure 2.3: Real and expected work plans for second semester.

This page is intentionally left blank.

Chapter 3

State of the Art

In this Chapter, we discuss the core concepts regarding the project, the most modern technology for the purpose today and related work in the area. All the information presented results from work of research through published articles, knowledge exchange and web searching.

First, the main purpose of Section 3.1 - Concepts is to introduce and provide a brief explanation about the core concepts to the reader. Second, Section 3.2 - Technologies, all the relevant technologies are analysed and discussed. In the final Section 3.3 - Related Work, published articles and posts of related work are presented and possible research directions are discussed.

3.1 Concepts

The following concepts represents the baseline to understand the work related to this research project. First an explanation of higher level of concepts that composes the title of this thesis are presented in Subsections 3.1.1 and 3.1.2. The following Subsections 3.1.3 to 3.1.5, aim to cover topics related to previous concepts: Distributed Tracing, Graphs and Time Series.

3.1.1 Microservices

The term "micro web services" was first used by Dr. Peter Rogers during a conference on cloud computing in 2005, and evolved later on to "Microservices" at an event for software architects in 2011, where the term was used to describe a style of architecture that many attendees were experimenting with at the time. Netflix and Amazon were among the early pioneers of microservices [8].

Microservices is "an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities" [1], [9].

This style of software development has a very long history and has being introduced and evolving due to software engineering achievements in the later years regarding cloud distributed computing infrastructures, Application Programming Interface (API) improvements, agile development methodologies and the emergence of the recent phenomenon of containerized applications. "A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another, communicating with others through an API" [10].

In Microservices, services are small, specifically calibrated to perform a single function, also each service is designed to be autonomous, resilient, minimal and composable. This framework brings a culture of rapid iteration, automation, testing, and continuous deployment, enabling teams to create products and deploy code exponentially faster than ever before [11].

Until the rising of Microservices based architecture, the Monolithic architectural style was the most used. This style has a the particularity of produce software composed all in one piece. All features are bundled, packaged and deployed in a single tier application using a single code base.

Figure 3.1 aims to give a comparison between both architectural styles, Monolithic and Microservices, and provide an insight about the differences between them.

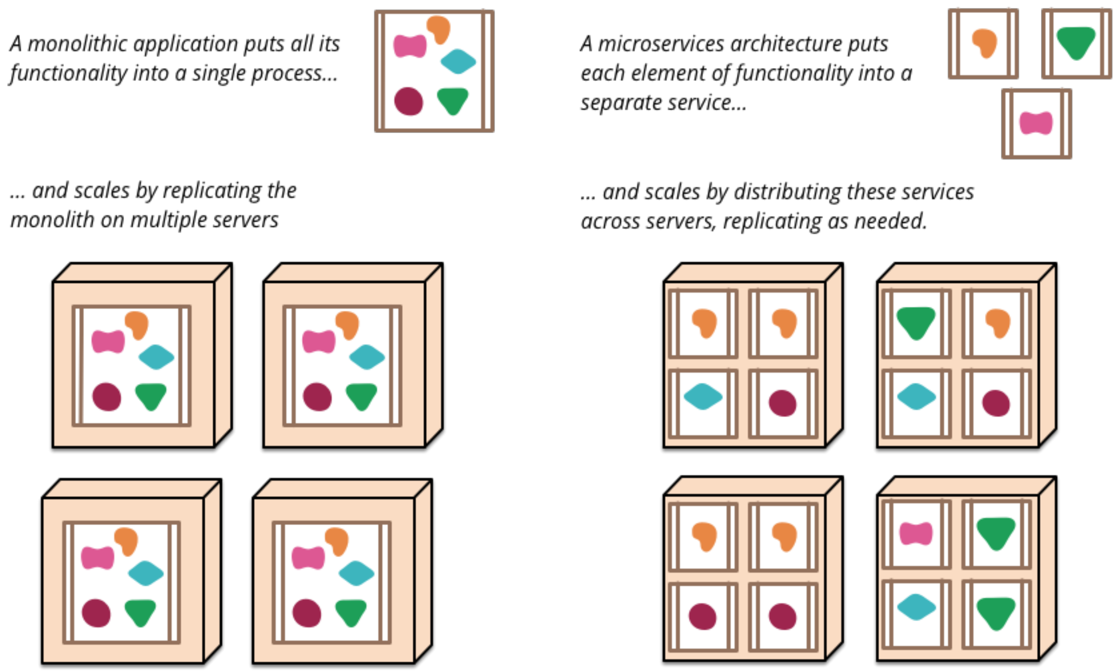

Figure 3.1: Monolithic and Microservices architectural styles [12].

Both styles presented have their own advantages and disadvantages. To briefly present some of them, two examples are provided, one for each architectural style. First example: if one team needs to develop a single process system, e.g., e-Commerce application, that authorizes customer, takes an order, check products inventory, authorize payment and ships ordered products. The best alternative is to use Monolithic architecture, because they can develop every feature in a single software package due to the application simplicity, however, if the client starts to demand hard changes and additional features in the solution, the code base may tend to increase into "out of control", leading to more challenging and time consuming changes. Second example, if one team needs to develop a complex and huge service that needs to scale, e.g., Video streaming service, the best alternative is to use Microservices architecture, because they can tackle the problem of complexity by decomposing the application into a set of manageable small services which are much faster to develop and test by individual organized teams, and thus, it will be easier to maintain the code base due to decoupling, however, it will be harder to monitor and manage the entire platform due to additional complexity associated with distributed systems.

Taking into consideration this increasing difficulty in monitoring and managing large Microservice based platforms, one must be aware and observe system behaviour to be able to control it. Therefore, in the next Subsection 3.1.2, the core concept of Observability and Controlling Performance is explained.

3.1.2 Observability and Controlling Performance

This Subsection aims to provide an introduction to some theory concepts about Observability and Performance Controlling, regarding distributed software systems.

Observability is a meaningfully extension of the word observing. Observing is "to be or become aware of, especially through careful and directed attention; to notice' '[13]. The term Observability comes from the world of engineering and control theory. Observability is not a new term in the industry, however it has gain more focus in the last years due to Development and Operations (DevOps) raising. It means by definition "to measure of how well internal states of a system can be inferred from knowledge of its external outputs" [14]. Therefore, if our good old software systems and applications do not adequately externalize their state, then even the best monitoring can fall short.

Controlling in control systems is "to manage the behaviour of a certain system" [15]. Controlling and Observability are dual aspects of the same problem [14], as we need to have information to infer state and be able take action. E.g., When observing an exponential increase in the Central Processing Unit (CPU) load, the system scales horizontally invoking more machines and spreading the work between them to easy handle the work. This is a clear and simple example that conjugates the terms presented, we have: values that are observed "Observability" and action that leads to system control "Controlling Performance".

When we want to understand the working and behaviour of a system, we need to watch it very closely and pay special attention to all details and information it provides. Microservice based systems produce multiple types of information if instrumented. These type of information are the ones mentioned in Chapter 1: Monitoring, Tracing and Logging. In this thesis, the goal is to use tracing data thus, this type of produced information is the one to focus.

In the next Subsection 3.1.3 - Distributed Tracing, the type of data mentioned before is presented and explained in detail.

3.1.3 Distributed Tracing

Distributed tracing [16] is a method that comes from traditional tracing, but in this case acts in a distributed system at the work-flow level. It is used to profile and monitor applications, especially those built using microservice architectures and, in the end, it can be used to help DevOps teams pinpoint where failures occur and what causes system problems.

From this concept, standards emerged, like the best-known OpenTracing [17]. The OpenTracing standard, follows the model proposed by Fonseca *et al.* [18], which defines traces as a tree of spans, which represent scopes or units of work (i.e., thread, function, service) and follows their executing through the system.

OpenTracing uses dynamic, fixed-width metadata to propagate causality between spans, meaning that each span has a *TraceID* common to all spans of that trace, as well as a *SpanID* and *ParentID* that are used to represent parent/child relationships between them [19].

The standard defines the format for spans and the semantic [20], [21] conventions for their content / annotations.

Figure 3.2 provides a clear insight about how spans are related to time and with each other.

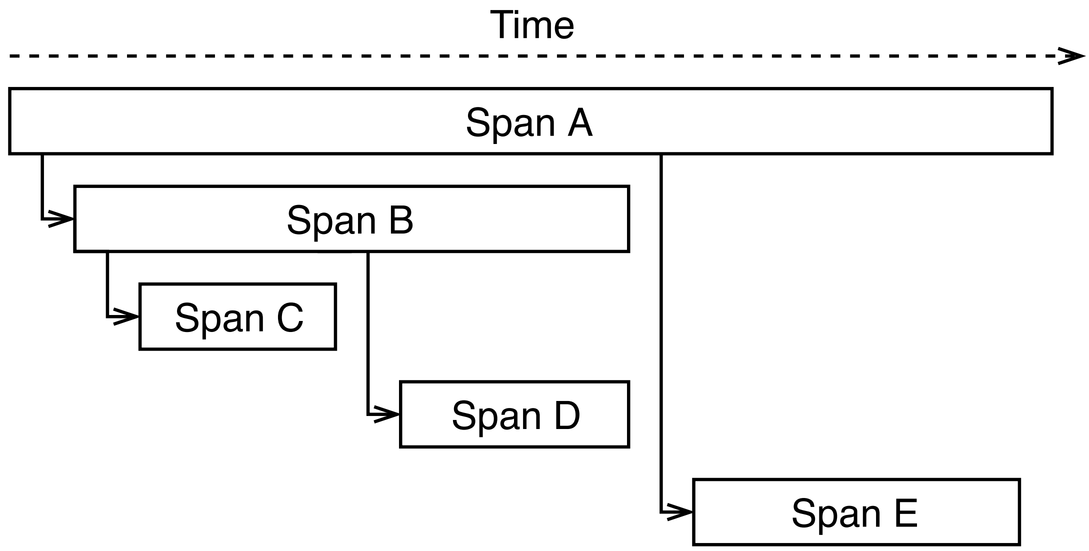

Figure 3.2: Sample trace over time.

In Figure 3.2 there are a group of five spans spread through time that represents a trace. A trace is a group of spans that share the same *TraceID*. A trace is a representation of a data/execution path in the system. A span represents the logical unit of work in the system. A trace can also be a span, if there is only one span presented in the trace. One span can cause another.

Causality relationship between spans can be observed in Figure 3.2, where “Span A” causes “Span B” and “Span E”, moreover, “Span B” causes “Span C” and “Span D”. From this we say that “Span A” is parent of “Span B” and “Span E”. Likewise, “Span B” and “Span E” are children of “Span A”. In this case, “Span A” does not have a parent, it is an “orphan span” and therefore, is the root span and the origin of this whole trace. Spans carry with them metadata like e.g., *SpanID* and *ParentID*, that allows to infer this relationships.

Disposition of spans over time is another clear fact that can be observed from the representation in Figure 3.2. Spans have a begin and an end in time. This causes them to have a duration. Spans are spread through time, however they usually stay inside parent boundaries, this means that the duration of a parent span always covers durations of their children. Considering a parent and a child spans, if they are related, the parent span always start before child span, also, the parent span always end after child span. Note that nothing prevents multiple spans to start in the same exact moment. Span also carry with them metadata like e.g., *Timestamp* and *Duration*, that allows to infer their position in time and when they end.

An example of a span can be an Hypertext Transfer Protocol (HTTP) call or a Remote Procedure Call (RPC) call. We may think of the following cases to define each operation inherent to each box presented in Figure 3.2: A - "Get user info", B - "Fetch user data from database", C - "Connect to MySQL server", D - "Can't connect to MySQL server" and E - "Send error result to client".

In the data model specification, the creators of OpenTracing say that: "with a couple of spans, we might be able to generate a span tree and model a directed graph of a portion of the system" [17]. This is due to the causal relationships they represent. Figure 3.3 provides an example of a span tree.

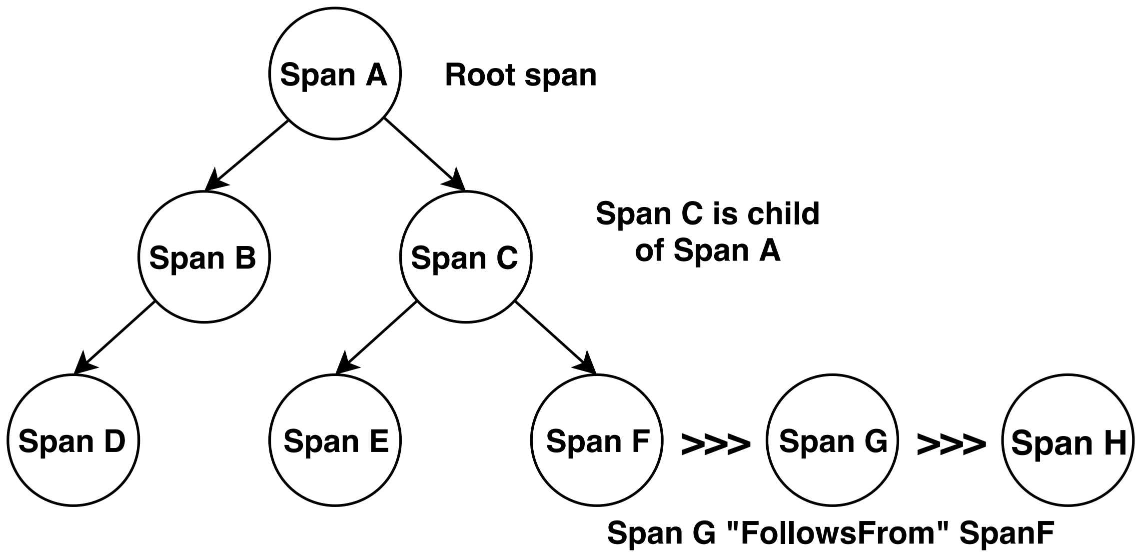

Figure 3.3: Span Tree example.

Figure 3.3 contains a span tree representation with a trace containing eight spans. As said before,every span must be a child of some other span, unless it is the root span, this is very clear in a span tree visualization with the usage of the root node. With this causal relationship, a path through the system can be retrieved. For example, if for example every span processes in a different endpoint represented by letters presented in the span tree, one may generate the request path: A —> B —> D. This means that our hypothetical request passed through machine A, B and D, or if it were services, the request passed from service A, to B and finally to D. From this, we can generate the dependency graph of the system (explained in the Subsection 3.1.4 - Graphs).

This type of data is extracted as trace files or streamed over transfer protocols like e.g., HTTP, from technologies like Kubernetes [22], OpenStack [23], and other cloud or distributed management system technologies that implements some kind of system or code instrumentation using, for example, OpenTracing [24] or OpenCensus [25]. Tracing contains some vital system details as they are the result of system instrumentation and therefore, this data can be used as a resource to provide observability over the distributed system.

As said before, from the causality relationship between spans we can generate a dependency graph of the system. The next Subsection 3.1.4 - Graphs aims to provide a clear understand of this concept and how they relate with distributed tracing.

3.1.4 Graphs

From distributed tracing we can be able to extract the system dependency graph from a representative set of traces. To introduce the concept of Graph, "A Graph is a set of vertices and a collection of directed edges that each connects an ordered pair of vertices" [26].

Taking the very common sense of the term and to provide notation, a graph, G, is an ordered pair $G = (V, E)$, where V are the vertices/nodes and E are the edges.

Graphs are defined by:

- Node: Are the entities in the graph. They can hold any number of attributes (key-value pairs) called properties. Nodes can be tagged with labels, representing their different roles in a domain. Node labels may also serve to attach metadata (such as index or constraint information) to certain nodes;
- Edge (or Relationships): provide directed, named, semantically-relevant connections between two node entities;
- Property: can be any kind of metadata attached to a certain Node or a certain Edge.

Also, there are multiple types of graphs, they can be:

1. Undirected-Graph: the set of edges without orientation between a pair of nodes;
2. Directed-Graph: the set of edges have one and only one direction between a pair of nodes;
3. Multi-Directed-Graph: multiple edges have more than one connection between a pair of nodes that represents the same relationship.

Figure 3.4 gives us a simple visual representation of what a graph really is for a more clear understanding.

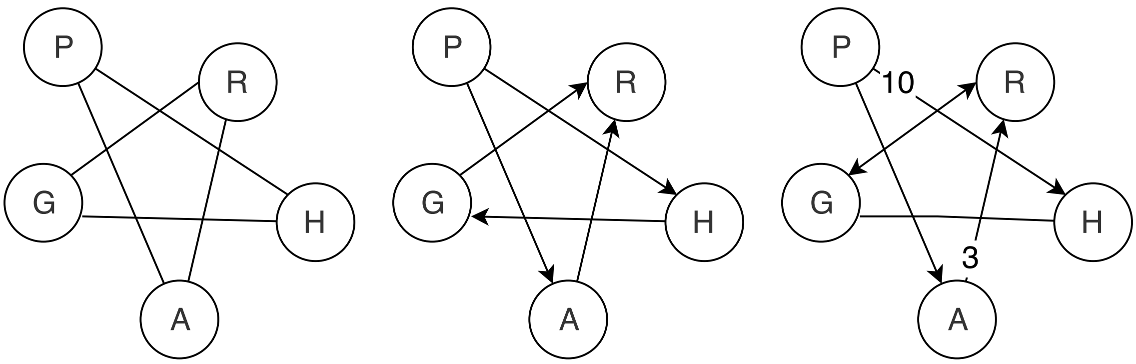

Figure 3.4: Graphs types.

In Figure 3.4 three identical graphs are presented and each one is composed by five nodes, however, they are not equal because each one has it own type. They belong respectively to each type enumerated above. From left to right, the first graph is a Undirected-Graph, the second one is a Directed-Graph and the last one is a Multi-Directed-Graph.

The last graph has some numbers in some edges. Every graph can have this annotations. These can provide some information about the connection between the pair of nodes. For example, in distributed systems context, if this graph represents our system dependency graph, and nodes H and P hypothetical services, the edge between them could represent calls between these two service and the notation number the number of calls with respect to the edge direction. Therefore, in this case, we would have 10 requests from incoming from P to H.

Figure 3.5 provides a clear insight about service dependency graphs.

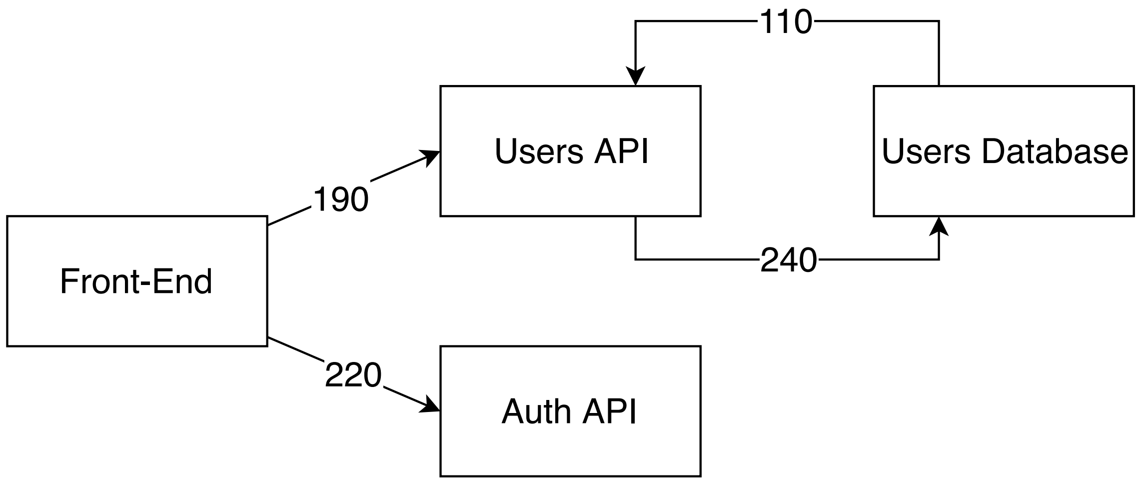

Figure 3.5: Service dependency graph.

In Figure 3.5, a representation of a service dependency graph is provided. Service dependency graphs are graphs of type Multi-Directed-Graph, because they have multiple edges with more than one direction between a pair of services(Nodes). In this representation, there are multiple services involved, each inside a box. The edges between boxes (Nodes), indicate the number of calls that each pair of services invoked, e.g., "Users API" called "Ùsers Database" 240 times. These dependency graphs gives the state of the system in a given time interval. This can be useful to study the changes in the morphology of the system, e.g., a service disappeared and a set of new ones appeared. Other interesting study could be the variation in the amount of call between services.

Graphs are a way to model and extract information from tracing data. Another interesting approach could be to extract metrics in time from tracing because traces and spans are spread in time, and they have information about the state of the system at a given instant. The next Subsection 3.1.5 - Time Series provides an introduction to a data representation model.

3.1.5 Time Series

Time-Series are a way of representing data as a time-indexed series of values. This kind of data is often arise when monitoring systems, industrial processes, tracking corporate business metrics or sensor measurements. Figure 3.6 provides a visual example of this way of data representation.

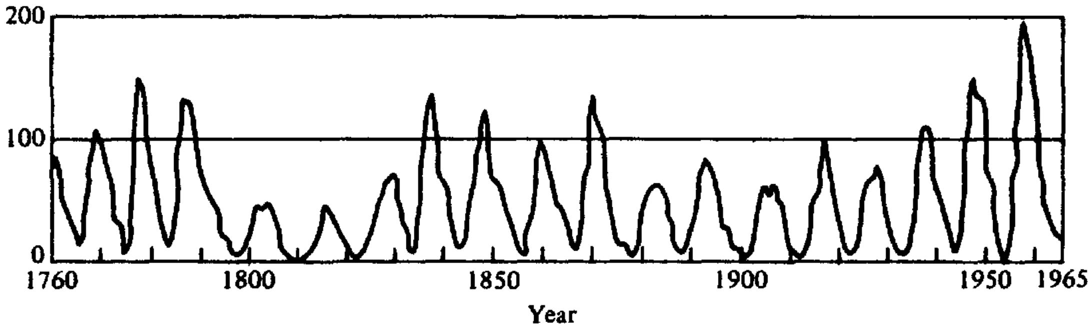

Figure 3.6: Time series: Annual mean sunspot numbers for 1760-1965 [27].

In Figure 3.6, Brillinger *D.* [27] presents a visual representation of a time-series as a collection of values in time. These values are measurements of sunspot means gathered from 1960-1965. In this case, measurements come from natural origin, however, one can perform observations of e.g., CPU load, system uptime / downtime and network latency.

As these processes are not random, autocorrelation can be exploited to extract insight from the data, such as predict patterns or detect anomalies. Therefore, time-series data can be analysed to detect anomalies present in the system. One way to do this is to look for outliers [28] in the multidimensional feature set. Anomaly detection in time series data is a data mining process used to determine types of anomalies found in a data set and to determine details about their occurrences. Anomaly detection methods are particularly interesting for our data set since it would be impossible to manually tag the set of interesting anomalous points. Figure 3.7 provides a simple visual representation of anomaly detection in time series data.

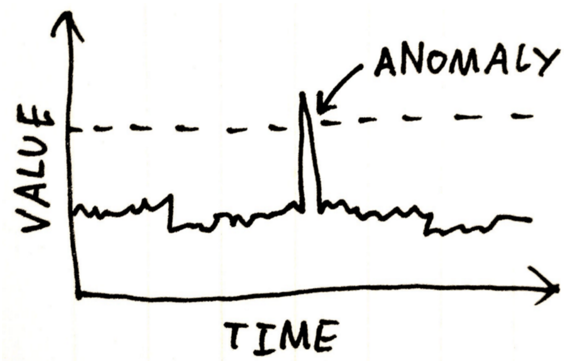

Figure 3.7: Anomaly detection in Time Series [29].

In Figure 3.7, there is a clear spike in values from this time series measurements. This can be declared an outlier because it is a strange value considering the range of remaining measurements and therefore, it is considered an anomaly. In this example, anomaly detection is easy to perform by a Human, however, in mostly cases nowadays, due to great variation of values and plethora of information that can be gathered, perform this detection manually is impracticable, thus automatic anomaly detection using Machine Learning techniques are used nowadays.

Anomaly detection in time series data is a data mining process used to determine types of anomalies found in a data set and to determine details about their occurrences. This auto anomaly detection method has lots of usage due to the impossible work of tag manually the interesting set of anomalous points. Auto anomaly detection has a wide range of applications such as fraud detection, system health monitoring, fault detection, event detection systems in sensor networks, and so on.

After explaining the core concepts, foundations for the work presented in this thesis, to the reader, technologies capable of handling this types of information are presented and discussed in next Section 3.2 - Technologies.

3.2 Technologies

In this section are presented technologies and tools capable of handling the types of information discussed in the previous Section 3.1 - Concepts.

The main tools covered are: 3.2.1 - Distributed Tracing Tools, for distributed tracing data handling, 3.2.2 - Graph Manipulation and Processing Tools and 3.2.3 - Graph Database Tools, for graph processing and storage, and 3.2.4 - Time-Series Database Tools, for time series value storage.

3.2.1 Distributed Tracing Tools

This Subsection presents the most used and known distributed tracing tools. These tools are mainly oriented for tracing distributed systems like microservices-based applications. What they do is to fetch or receive trace data from this kind of complex systems, treat the information, and then present it to the user using charts and diagrams in order to explore the data in a more human-readable way. One of the best features presented in this tools, is the possibility to perform queries on the tracing (e.g., by trace id and by time-frame). Table 3.1 presents the most well-known open source tracing tools.

In Table 3.1, we can see that these two tools are very similar. Both are open source projects, allow docker containerization and provide a browser ui to simplify user interaction. Jaeger was created by Uber and the design was based on Zipkin, however, it does not provide much more features. The best feature that was released for Jaeger in the past year was the capability of perform trace comparison, where the user can select a pair of traces and compare them in terms of structure. This is a good effort in additional features, but it is short in versatility because we can only compare a pair of traces in a "sea" of thousands, or even millions.

These tools aim to collect trace information and provide a user interface with some query capabilities for DevOps to use. However they are always focused on span and trace lookup and presentation, and do not provide a more interesting analysis of the system, for example to determine if there is any problem related to some microservice presented in the system. This kind of work falls into the user, DevOps, as they need to perform the tedious work of investigation and analyse the tracing with the objective of find anything wrong with them.

This kind of tools can be a good starting point for the problem that we face, because they already do some work for us like grouping the data generated by the system and provide a good representation for them.

In next Subsection 3.2.2, graph manipulation and processing tools are presented and discussed.

Table 3.1: Distributed tracing tools comparison.

	Jaeger [30]	Zipkin [31]
Brief description	Released as open source by Uber Technologies and is used for monitoring and troubleshooting microservices-based distributed systems. Was inspired by Zipkin.	Helps gather timing data needed to troubleshoot latency problems in microservice architectures and manages both the collection and lookup of this data. Zipkin's design is based on the Google Dapper paper.
Pros	OpenSource; Docker-ready; Collector interface is compatible with Zipkin protocol; Dynamic sampling rate; Browser UI.	OpenSource; Docker-ready; Allows lots of span transport ways (HTTP, Kafka, Scribe, AMQP); Browser UI.
Cons	Only supports two span transport ways (Thrift and HTTP).	Fixed sampling rate.
Analysis	Dependency graph view; Trace comparison (End 2018).	Dependency graph view.
Used by	Red Hat; Symantec; Uber.	AirBnb; IBM; Lightstep.

3.2.2 Graph Manipulation and Processing Tools

Distributed tracing is a type of data produced by Microservice based architectures. This type of data is composed by traces and spans. With a set of related spans, a service dependency graph can be produced. This dependency graph is a Multi-Directed-Graph, as presented in Subsection 3.1.4. Therefore, with this data at our disposal, there is the need of a graph manipulation and processing tool.

In this Subsection, the state of the art about graph manipulation and processing tools is presented. Graphs are non-linear data structure representations consisting of nodes and edges. Nodes are sometimes also referred to as vertices and edges are lines or arcs that connect any pair of nodes in the graph. This data structure takes some particular approaches when handling their contents, because there are some special attributes related. For example, perform the calculation of the degree of some node – degree of a node is the number of edges that connect to the node itself; Calculate how many nodes entered and exited the graph by comparing it to another one; Know the difference in edges between two distinct graphs [32].

Taking into consideration this data structure, the particularities involved and the need to use graphs to manipulate service dependencies, frameworks with features capable of handling and retrieving graphs are a need. Therefore, Table 3.2 presents a comparison of the main tools available at the time for graph manipulation and processing.

Table 3.2: Graph manipulation and processing tools comparison.

	Apache Giraph [33]	Ligra [34]	NetworkX [35]
Description	An iterative graph processing system built for high scalability. Currently used at Facebook to analyse the social graph formed by users and their relationships.	A library collection for graph creation, analysis and manipulation of networks.	A Python package for the creation, manipulation, and study of structure, dynamics, and functions of complex networks.
Licence [36]	Free Apache 2.	MIT.	BSD - New License.
Supported languages	Java and Scala.	C and C++.	Python.
Pros	Distributed and very scalable; Excellent performance – Process one trillion edges using 200 modest machines in 4 minutes.	Handles very large graphs; Exploit large memory and multi-core CPU – Vertically scalable.	Good support and very easy to install with Python; Lots of graph algorithms already implemented and tested.
Cons	Uses “Think-Like-a-Vertex” programming model that often forces into using suboptimal algorithms, thus is quite limited and sacrifices performance for scaling out; Unable to perform many complex graph analysis tasks because it primarily supports Bulk synchronous parallel.	Lack of documentation and therefore, very hard to use; Does not have many usage in the community.	Not scalable (single-machine); High learning curve due to the maturity of the project; Begins to slow down when processing high amount of data – 400.000+ nodes.

Table 3.2 presents some key points to consider when choosing a graph manipulation and processing tool.

First, one aspect to be considered when comparing them is the scalability and performance that each provide. Apache Giraph is the best tool in this field, since it is implemented with distributed and parallel computation, which allows it to scale to multiple-machines, sharing the load between them, and processing data large quantities of data in less time than the remaining presented tools. On the opposite side, NetworkX, only works in a single-machine environment which does not allow it scale to multiple-machines. Ligra, like the previous tool, works in a single-machine environment, however it benefits from vertical scale on a single-machine, which allows to exploit multi-core CPU and large memory. NetworkX and Ligra are tools that can present a bottleneck in a system where the main focus is to handle large amounts of data in short times.

Secondly, another aspect to be considered is the support and quantity of implemented graph algorithms available on the frameworks. NetworkX have advantages in this aspect, because it contains implementation of the majority graph algorithms defined and studied in graph and networking theory. Also, due to project maturity, it has a good documentation support from the community who keeps all the information updated. Ligra framework has lack of documentation, which causes tremendous difficulty for developers to use and know what are the implemented features. Apache Giraph, does not support a large set of graph processing algorithms due to implementation constraints.

Figure 3.8 gives a clear insight when comparing these tools from two features – scalability / performance against implementation of graph algorithms.

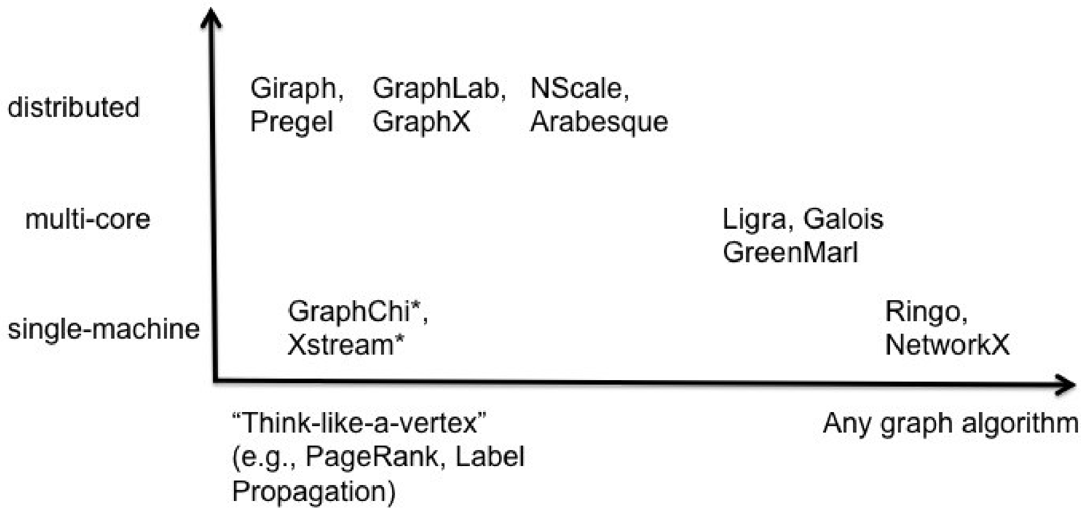

Figure 3.8: Graph tools: Scalability vs. Algorithm implementation [37].

In Figure 3.8 we can observe tools disposition regarding the two aspect key points explained before. This figure contains all tools presented over two featured axis: one for scalability and the other for implementation of graph algorithms. Tools placement in this chart proves and reinforces the comparison presented before. Apache Giraph and NetworkX are placed in the edges of these features, which means that Apache Giraph can be found in the upper left region of the chart – highly distributed but minimally in graph algorithms implementation –, and NetworkX is in the lower right region – minimally distributed but highly in graph algorithms implementation.

After discussing tools capable of manipulate and process graphs, their storage is a need for later usage. Graph Database (GDB) storage technologies are presented in next Subsection 3.2.3 - Graph Database Tools.

3.2.3 Graph Database Tools

Graph databases represent a way of persisting graph information. After having instantiated a Graph, processed it in volatile memory, they can be stored in persistent memory for later use. To do this one can use a GDB. A GDB is "a database that allows graph data storing and uses graph structures for semantic queries with nodes, edges and properties to represent them" [38].

Based upon the concept of a mathematical graph, a graph database contains a collection of nodes and edges. A node represents an object, and an edge represents the connection or relationship between two objects. Each node in a graph database is identified by a unique identifier that expresses key value pairs. Additionally, each edge is defined by a unique identifier that details a starting or ending node, along with a set of properties. Graph databases are becoming popular due to Machine Learning and Artificial Intelligence grows, since a number of Machine Learning algorithms are inherently graph algorithms [39].

Furthermore, in this research service dependency graphs are highly used, thus the need to use a GDB. Table 3.3 contains the most well-known GDB.

Table 3.3: Graph databases comparison.

	ArangoDB [40]	Facebook TAO [41]	Neo4J [42]
Description	A NoSQL database that uses a proper query language to access the database.	TAO, "The Associations and Objects", is a proprietary graph database, developed by Facebook, used to store the social network.	The most popular open source graph database, completely open to the community.
Licence [36]	Free Apache 2.	Proprietary.	GPLv3 CE.
Supported languages	C++; Go; Java; JavaScript; Python and Scala.	Go; Java; JavaScript; Python and Scala.	Java; JavaScript; Python and Scala.
Pros	Multi data-type support (key/value, documents and graphs); Allows combination of different data access patterns in a single query; Supports cluster deployment.	Low latency ($= 100ms$); Accepts millions of calls per second; Distributed database.	Supports ACID(Atomicity, Consistency, Isolation, Durability) [43]; Most popular open source graph database.
Cons	High learning curve due to AQL (Arango Query Language); Has paid version with high price tag.	Not accessible to use.	Not able to scale horizontally.

From Table 3.3 we can notice that the state of the art for GDB is not very pleasant, because the interest for this type of databases has began in the later years due to artificial intelligence and machine learning trends. Therefore, the offer presented in the field are limited.

Back in time, when social network tendency emerged, the development of this type of databases raised, and the most powerful technologies for graph storage where developed in closed source. One example is Facebook TAO database presented in Table 3.3, a database developed by the company to support the entire social network, storing users in nodes and their relationships in edges. This database is described by having very low latency, which stands for high response time however, just a few scientific papers were found [41], [44].

The remaining tools presented are available for usage. ArangoDB has multi data-type support, which means that a wider type of data structures are supported for storing in nodes and edges metadata. Also, it supports scalability through cluster deployment, however, this feature is only available in paid versions – Arango SmartGraphs storage improves the writing of graph in distributed systems environment [45]. The biggest disadvantage of this database is the high learning curve associated with the usage of AQL (Arango Query Language), however, this disadvantage can be surpassed by using provided API clients with the trade-off of loosing some control.

Neo4J is the most accepted GDB by the open source community. This GDB has increased in popularity in the past years due to simplicity and easy support [46]. Trade-offs from this database consists in lack of support for scalability, which means that this database can only run on a single-machine environment, however, there are some users reporting that they were able to perform implementations and surpass the lack of support for horizontal scaling, but this is not tested [47].

Choosing a graph database can be hard because these tools are growing and the tendency for changes in features and tooling support is very high, however, the decision falls on the question of easy of usage and horizontal scalability. This means that ArangoDB is a database more advised for big projects, where the size of graphs to store may surpass the limit of a single-machine, and Neo4J for simpler projects, where the focus are functionality testing and prototyping, and graph storage represents a side concern.

Next Subsection 3.2.4 - Time-Series Database Tools covers the state of the art for tooling capable of storage values based in time.

3.2.4 Time-Series Database Tools

In this Subsection, tools for storing time-indexed series of values are presented. This type of data is a need for this research due to the tight relation between distributed tracing and time, as explained in Subsections 3.1.3 and 3.1.5. Also, service dependency graphs, as a representation of the system at a given time, can contain valuable information for monitoring Microservice systems. For this purpose, Time Series Database (TSDB) are databases capable of storing time-series based values.

A TSDB is “A database optimised for time-stamped or time series data like arrays of numbers indexed by time (a date time or a date time range)” [48]. These databases are natively implemented using specialised time-series theory algorithms to enhance their performance and efficiency, due to widely variance of access possible. The way this databases use to work on efficiency is to treat time as a discrete quantity rather than as a continuous mathematical dimension. Usually a TSDB allows operations like create, enumerate, update, organise and destroy various time series entries in short access times.

This type of database is growing in usage and popularity because of Internet of Things (IoT) trend. Discussions in this area have increased over the past few years, and is expected that it keeps increasing, due to Ubiquitous Computing – Raise of omnipresent and universal technologies. At the same time, TSDB grows with this IoT tendency, because data mining from sensor spread geographically and sensors gather information through measurements in specific points in time. This information are usually stored in TSDB [49].

Table 3.4 presents a comparison between two TSDB: *InfluxDb* and *OpenTSDB*.

Table 3.4: Time-series databases comparison.

	InfluxDB [50]	OpenTSDB [51]
Description	An open-source time series database written in Go and optimised for fast, high-availability storage and retrieval of time series data in fields such as operations monitoring, application metrics, Internet of Things sensor data, and real-time analytics's.	A distributed and scalable TSDB written on top of HBase; OpenTSDB was written to address a common need: store, index and serve metrics collected from computer systems (network gear, operating systems and applications) at a large scale, therefore, making this data easily accessible and displayed.
Licence [36]	MIT.	GPL.
Supported languages	Erlang, Go, Java, JavaScript, Lisp, Python, R and Scala.	Erlang, Go, Java, Python, R and Ruby.
Pros	Scalable in the enterprise version; Outstanding high performance; Accepts data from HTTP, TCP, and UDP protocols; SQL like query language; Allows real-time analytics's.	Massively scalable; Great for large amounts of time-based events or logs; Accepts data from HTTP and TCP protocols; Good platform for future analytical research into particular aggregations on event / log data; Does not have paid version.
Cons	Enterprise high price tag; Clustering support only available in the enterprise version.	Hard to set up; Not a good choice for general-purpose application data.

From Table 3.4, we can notice some similarities between these two TSDB databases. Both TSDB are capable scalable and accept HTTP and TCP transfer protocols for communication. InfluxDB and OpenTSDB are two open source time-series databases, however, the first one, InfluxDB, is not completely free, as it has an enterprise paid version, which is not very visible in the offer. This enterprise version offers, clustering support, high availability and scalability [52], features that OpenTSDB offer for free. In terms of performance, InfluxDB surpasses and outperforms OpenTSDB in almost every benchmarks [53]. OpenTSDB has the benefits of being completely free and support the most relevant features, however it is very hard to set up and to develop for this database.

In the end, both TSDB are bundled with good features, and the decision falls into how much performance is needed when choosing one. If the need is performance and access to the database in short amounts of time, with low latency responses, InfluxDB is the way to go, by other way, if there no restriction about the performance needed to query the database and money is a concern, the choice should be OpenTSDB.

Tooling for this project is presented. We have covered the most used technologies and core concepts in related to the field of tracing Microservices. Next Section 3.3 - Related Work, will cover the related work performed in this area. Some ideas, approaches and developed solutions will be discussed.

3.3 Related Work

This section aims to present the related work in the field of distributed tracing data handling and analysis. It is divided in three Subsections: first, 3.3.1 - Mastering AIOps, which covers a work carried out by Huawei, that uses machine learning – deep learning – methods to analyse data from distributed traces. Secondly, 3.3.2 - Anomaly Detection using Zipkin Tracing Data, a work of performed by Salesforce with the objective of analyse tracing from a distributed tracing tool. Finally, 3.3.3 - Analysing distributed trace data, a work by Pinterest, where the objective is to study latency in tracing data.

3.3.1 Mastering AIOps

Distributed tracing has only started to gain widespread acceptance in the industry recently, as a result of new architectural and software engineering practices, such as cloud-native, fine-grained systems and agile methodologies. Additionally, the increase in complexity resulting from the rise of web-scale distributed applications is a recent phenomenon. As a consequence of its novelty, there has been little research in the field so far.

A recent example, AIOps, an application of Artificial Intelligence to operations [54] was introduced in 2016 [55]. This trend aims to use Artificial Intelligence for IT Operations in order to develop new methods to automate the enhance IT Operations. Driving this "revolution" are the following points:

- First there is the additional difficulty of manually managing distributed infrastructures and system state;
- Secondly, the amount of data that has to be retained is increasing, creating a plethora of problems to the operators handling it;
- Third, the infrastructure itself is becoming more distributed across geography and organizations, as evidenced by trends like cloud-first development and fog computing;
- Finally, due to the overwhelming amount of new technologies and frameworks, it is an herculean task for operators to keep in pace with the new trends.

The work performed and presented by Huawei, entitled Mastering AIOps with Deep Learning, Time-Series Analysis and Distributed Tracing [56], aims to use distributed tracing data and aforementioned technologies to detect anomalous tracing. The proposed method encodes the traces and trains a deep learning neural network to detect significant differences in tracing. This is a very perceptive approach, taking into account the amounts of data that is needed to analyse, however is limited to classifying a trace as normal or abnormal, losing detail and interpretability i.e., no justification for the classification.

3.3.2 Anomaly Detection using Zipkin Tracing Data

Tooling in this field are not taking the expected relevance. Their usage is starting in industry and production environments involving distributed systems, however, the concerns in are not well aligned with the needs of operators, and this leads to increasing effort when monitoring large scale and complex architectures, such as Microservices.

In a post from Salesforce, a work of research about using tracing data gathered by Zipkin, to detect some anomalies in a Microservice based system [57]. At Salesforce, Zipkin is used to perform distributed tracing for Microservices, collecting traces from their systems and providing performance insights in both production monitoring and pre-production testing. However, the current Zipkin open source instrumentation and UI offers only primitive data tracing functionality and does not have in-depth performance analysis of the span data. The focus on their work was to detect and identify potential network bottlenecks and microservices performance issues.

The approach carried out was to implement scripts using that used Python AI packages, with the objective of extracting values from their network of services, namely service dependency graph, in order to identify high traffic areas in the network. The values that were extracted were the number of connections from each service, which means, the degree of the service at specific times. This allows to notice which services are establishing more connections with other services.

From this approach, it was possible to visualize the high traffic areas within the production network topology. Therefore, they have identified services with the most connections. This finding was an helpful feedback for service networking architects that designed those microservices. Those services, identified with too many connections, may potentially become choking points in the system design. If one of the services fail, a huge impact on a large number of depending services occur. Additionally, there could be also potential performance impacts in the system since a large number of services depending on them. Those are valuable information for system designers and architect to optimize their designs.

The conclusions from Salesforce research identified that, with Zipkin tracing data, it is possible to identify network congestion, bottlenecks, efficiencies and the heat map in the production network. However, this tool does not provide analysis of tracing data at this level. This was the main conclusion and possible working direction from this research: "features like the ones presented, can be added to Zipkin or other distributed tracing tool product line, including UI and dashboards. Capabilities like daily metrics or correlation between microservices load and latency, able to generate alerts if bottleneck or heat map is identified, should be added" [57].

3.3.3 Analysing distributed trace data

At Pinterest, the focus were to research for latency problems in their Microservices solution. Pinterest claims to have tens of services and hundreds of network calls per-trace. One big problem identified at start is the huge difficulty of looking to trace data due to overwhelming quantity of information – "thousands of traces logged each minute (let alone the millions of requests per minute these traces are sampled from)".

Pinterest felted the problem of monitoring Microservices early due to their service popularity in the past years. With this popularity, systems usage have increased significantly. This lead them to take action and create their closed source distributed tracing analysis tool called "Pintrace Trace Analyser" [58].

This tool gathers tracing data from Distributed Tracing Tools, more precisely from Zipkin, and processes a sample of these tracing to detect mainly latency problems in the service dependency network. Looking at stats from thousands of traces over a longer period of time not only weeds out the outliers/buggy traces, but provides a holistic view of performance.

The conclusions from Pinterest, where that there is a great need to develop tooling for distributed tracing analysis, with the main objective of ease the life of operators. The following points were considered:

1. Automatically generate reports so engineers can easily check the status of each deployment;
2. Setting up alerts for when latency or number of calls hits a certain threshold.

3.3.4 Research possible directions

In this Subsection, the related work previously presented will be analysed, and from this, some possible directions of research will be considered. These considerations will be further discussed in Chapter 4 - Research Objectives and Approach.

One thing to notice from the related work presented is that there is few research accomplished in the area and trace tooling development, however, these works are from the past year and the tendency is to increase in the following years. Enlargement and usage of distributed systems are fuel to feed the need of research in this field and develop new methodologies and tools to monitor and control operations.

From the first work presented, Mastering AIOps, some final results and conclusions were provided. They point out that the benefits of this approach were: first, very high accuracy in detection $99,7\%$, and secondly, extremely fast detection in $O(n)$ time. However, some limitations involving requiring very long training times for long traces (with decent machines) were noted. Also, improvements were pointed: truncate traces, to lower the quantity of tracing and therefore, summarizing traces.

The second work presented, Anomaly Detection using Zipkin Tracing Data, point down the lack of features in the existing tools. These features include automatic anomaly detection using distributed tracing data. The main idea consists in extending functionality presented in this tools, to provide autonomous anomaly detection and alerting based on information presented in tracing from services.

In third work presented, Analysing distributed trace data, crucial points considered were to represent autonomous generation of reports, allowing operators to check the status of deployments, and therefore, providing more control over the system regarding detection of anomalous values in service latency.

Finally, from the multiple related work presented the final assumptions for possible research directions in this field are:

- Focus on the most important traces, reducing the quantity of tracing;
- Develop new methods that leverage features of existing distributed tracing tools;
- Automate the detection of anomalies presented in distributed systems;

After providing the state of the art for this research to the reader, next Chapter 4 - Research Objectives and Approach will cover the objectives of this research, the approach used to tackle the problem and the compiled research questions.

This page is intentionally left blank.

Chapter 4

Research Objectives and Approach

REVIEW THIS CHAPTER; Think about spliting it in two; Refer that the origin of the work comes from Huawei and professor JC is the client;

This chapter presents the research objectives and approach used in this thesis. We will start to discuss how we faced the problem, and what were the main difficulties that we found when handling this kind of problem as well as the ways we have taken to deal with it.

The debugging process in distributed systems and microservice based systems is not an easy task to perform, because of the way the system is designed using this kind of architecture style, as explained in the subsection 3.1.1 - Microservices. Reasoning about concurrent activities of system nodes and even understanding the system's communication topology can be very difficult. A standard approach to gaining insight into system activity is to analyse system logs, but this task can be very tedious and complex process. The main existing tools are the ones presented in 3.2.1, but they only do the job of gather and present the information to the user in a more gracefully way, however they rely on the user perception to do the search to find issues that exist in their platform by performing queries to the spans and trace data. So, with this problem ahead, we started looking for the needs of Sysadmins and/or Development and Operations (DevOps) when they wanted to scan and analyse their system searching for issues.

To do this and narrow the problem we were facing, we decided to talk with some DevOps personal and expose them the situation, with the objective to gather their main needs and ideas in mind, we putted ourselves in their perspective when talking to them to try and find what are the main difficulties when they perform their search for issues in the system in a *"As a DevOps i want to..."* situation. The kind of questions that were placed were like: *"What are the most common issues?"*, *"What are the variables involved in this kind of issues?"* and *"What are the correlations between this variables and the most common issues?"*. From this discussions and conversations emerged the following eight core questions:

1. What is the neighbourhood of one service?
2. Is there any problem (Which are the associated heuristics)?
3. Is there any faults related to the system design/architecture?
4. What is the root problem, when A, B, C services are slow?
5. How are the requests coming from the client?

6. How endpoints orders distributions are done?

7. What is the behaviour of the instances?

8. What is the length of each queue in a service?

The next step was to work on the questions presented above. We decided to split them in more concise questions, refine and filter the most relevant to define our objective, and after that, check with someone involved in the DevOps field if the final questions represent some of their needs. First, to handle the information presented in this eight initial questions, we decided to create what we called a "Project Questions Board". This board consists on a Kanban [59] style board present in the project git repository, were everyone involved in the project could access and modify it. The board was defined with four lanes: "To refine", "Interesting", "Refined" and "Final Questions", and the process were to cycle the questions through every lane, generating new ones and filtering others. After this, and to check if the final questions were really some that represented the needs of a DevOps, some colleagues that work directly in the field were contacted and the questions were exposed to them. In the end, the ten questions that were produced in the final lane represented right what are some of their needs. The final questions, their corresponding description**(D)** and explanation of the expected work**(W)** that must be performed to each one are exposed bellow:

1. What is the neighbourhood of one service, based on incoming requests?

- D. The neighbourhood of one service is a very important information to know due to the simple fact that it represents the interaction between the microservices. The incoming requests can map the interactions between the microservices and with this kind of information, we can check and analyse the service dependencies.
- W. Implies generate a graph, based on the spans and traces, using the outgoing connections, from a certain node, that are correlated with the incoming connection(s).

2. What is the neighbourhood of one service, based on outgoing requests?

- D. Similar to the previous question, but this time, instead of incoming requests we focus on the outgoing requests of one service.
- W. Implies generate a graph, based on the spans and traces, using the incoming connections, from a certain node, that are correlated with the outgoing connection(s).

3. How endpoints orders distributions are done, when using a specific endpoint?

- D. The distributions of microservices in a system allows us to understand if a certain endpoint groups with others or if it is an isolated service, which stands for its relevance to the whole system.
- W. Implies generate a graph, based on the spans and traces, then calculate the degree of a certain node that represents the endpoint, to finally check if it is an isolated, a leaf or a dominating (high or low depending on the degree of the other degrees) endpoint.

4. How requests are being handled by a specific endpoint?

D. This question has the objective of analyse the status of the requests that arrive or depart from a specific endpoint. This status represents if the requests was well succeed or not.

W. Implies to analyse the data from the requests that pass through a specific endpoint. Based on the annotations presented in the spans and traces, we are able to check if the requests are resulting in success or in error.

5. Which endpoints are the most popular?

D. The popularity of a certain endpoint is very important because it represents the importance of this endpoint to the system.

W. Implies to retrieve the most popular service, based on the spans and traces, and get the services with more incoming connections sorted by the number of incoming connections.

6. Is there any problem related to the response time?

D. Response time is must watch variable because, for example, strange high values may represent a problem in the system performance.

W. Implies to get the response time of every trace (difference between end and start time of every span in the trace) and then calculate and store some measurements like the average time, the maximum time, the minimum time and variance. After having some stored values, the system must perform calculations and check if there is too much disparity between them to determine if there is a problem in the response time.

7. Is there any problem related to the morphology?

D. The morphology of the system allows us to understand if some endpoints are common in the system (they usually exist), or if they only are instantiated in specific situations.

W. Implies generate multiple graphs, based on a certain group of spans and traces that are contained in a certain time interval. Then we need to store the graphs gradually using some graph storing mechanism to perform the difference of subsequent stored graphs. This result of the difference between graphs must be stored in a time-series storing mechanism, to be accessed later and determine if there were hard changes that could lead to morphology problems in the system thought time.

8. Is there any problem related to the entire work-flow of (one or more) requests?

D. Analyse the request work-flow through the system is a good practice, as it represents the interaction triggered by the request in the system and its resulting behaviour. This can lead to find out if the system has cycles and if they are normal or represent a problem to solve.

W. Implies to generate the graph of the system, identify the path of some request(s) in the system and then perform the calculation to verify and identify if there were cycles presented in the graph involved in the path of the request(s). The results of this calculations must be stored in a time-series storing mechanism, to be accessed later and determine if this cycles are normal, or if they represent a problem related to the request(s) work-flow, based on the kind of request.

9. Is there any problem related to the occupation/load of a specific endpoint?

D. The occupation/load of a specific endpoint in this case is represented by the number of requests in queue of a specific endpoint. This value is very important because it represents if the endpoint is in overflow or not.

W. Implies to get the number of requests in queue of a specific endpoint and then calculate and store some measurements like the average, maximum, minimum and variance of the number of requests in queue. After having some stored values, the system must perform calculations and check if there is too much disparity between them to determine if there is a problem in the occupation/load.

10. Is there any problem, related to the number/profile of the client requests?

D. The number of client requests and their corresponding profile represents the behaviour of clients when using the system. This can be used to identify problems of bad system usage from the clients, like for example a DDoS (Denial Of Service).

W. Implies calculate the number of accesses to the system, based on the spans and traces annotated with client requests of a certain time interval, and store the calculations for every node. After having some stored values, the system must determine the level regions of access based in the available data (profile of requests, ex.: high, moderate and low), and check if there were to much requests outside of the defined level regions.

These final questions, with a slight reformulation, could be exported to high level of abstraction functional requirements of the monitoring tool that we want to develop. After having this questions, we decided to study the current state of art to check how things are done nowadays regarding this subject and we found that some tools perform the process of convert spans and traces to a graph, that represents the system at that current time interval, however they do not perform any kind of analysis and study over the span tree and the graph after that[60].

Considering this, what we decided to do was to develop a simple prototype tool to test some state of the art tools. What we were able to achieve was to do the reconstruction of the graph, using our own data (this data was provided by Prof. Jorge Cardoso, representing an approximate two hour collection of spans and traces, about 400.000 spans, generated by one of their clusters). At this point, and since we have already held the hands-on of some tools at the moment, we were ready to start and think about the solution we need to build. Therefore, we decided to specify the solution, and considered to build a monitoring tool named by ourselves, *Graphy*.

In a very briefly explanation, we want that *Graphy* be able to calculate relevant metrics from the span trees and the generated graphs, and to work with this kind of metrics to perform the system analysis and answer the questions exposed above. To perform some of this work, it will be resourcing to machine learning algorithms that we will need to study in parallel with the implementation, as we cannot predict what we might encounter when retrieving the metrics at the time. The machine learning algorithms are to process the metrics and perform some deductions regarding the system behaviour over the time.

References

[1] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, today, and tomorrow", in *Present and Ulterior Software Engineering*, Cham: Springer International Publishing, 2017, pp. 195–216, ISBN: 9783319674254. DOI: 10.1007/978-3-319-67425-4_12. [Online]. Available: https://hal.inria.fr/hal-01631455.

[2] P. Di Francesco, I. Malavolta, and P. Lago, "Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption", in *Proceedings - 2017 IEEE International Conference on Software Architecture, ICSA 2017*, 2017, pp. 21–30, ISBN: 9781509057290. DOI: 10.1109/ICSA.2017.24. [Online]. Available: http://cs.gssi.infn.it/ICSA2017ReplicationPackage.

[3] J. Joyce, G. Lomow, K. Slind, and B. Unger, "Monitoring distributed systems", *ACM Transactions on Computer Systems*, vol. 5, no. 2, pp. 121–150, Mar. 1987, ISSN: 07342071. DOI: 10.1145/13677.22723. [Online]. Available: http://portal.acm.org/citation.cfm?doid=13677.22723.

[4] S. P. R. Janapati, *Distributed Logging Architecture for Microservices*, 2017. [Online]. Available: https://dzone.com/articles/distributed-logging-architecture-for-microservices.

[5] OpenTracing.io, *What is Distributed Tracing?* [Online]. Available: https://opentracing.io/docs/overview/what-is-tracing/.

[6] *Huawei Cloud Platform.* [Online]. Available: https://www.huaweicloud.com/.

[7] *GanttProject.* [Online]. Available: https://www.ganttproject.biz/ (visited on 11/29/2018).

[8] Laura Mauersberger, *Microservices: What They Are and Why Use Them.* [Online]. Available: https://blog.leanix.net/en/a-brief-history-of-microservices (visited on 06/05/2019).

[9] C. Richardson, *Microservices Definition.* [Online]. Available: https://microservices.io/ (visited on 10/17/2018).

[10] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, *Cloud Container Technologies: a State-of-the-Art Review*, 2017. DOI: 10.1109/TCC.2017.2702586. [Online]. Available: http://ieeexplore.ieee.org/document/7922500/.

[11] S. Newman, *Building Microservices: Designing Fine-Grained Systems.* 280, ISBN: 978-1-491-95035-7. [Online]. Available: http://ce.sharif.edu/courses/96-97/1/ce924-1/resources/root/Books/building-microservices-designing-fine-grained-systems.pdf.

[12] M. Fowler and J. Lewis, *Microservices, a definition of this architectural term*, 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html (visited on 01/07/2018).

[13] *Observing definition.* [Online]. Available: https://www.thefreedictionary.com/observing (visited on 10/13/2018).

[14] Peter Waterhouse, *Monitoring and Observability — What's the Difference and Why Does It Matter? - The New Stack.* [Online]. Available: https://thenewstack.io/monitoring-and-observability-whats-the-difference-and-why-does-it-matter/ (visited on 06/06/2019).

[15] Wikipedia, *Control system.* [Online]. Available: https://en.wikipedia.org/wiki/Control%7B%5C_%7Dsystem (visited on 01/02/2018).

[16] R. R. Sambasivan, I. Shafer, J. Mace, B. H. Sigelman, R. Fonseca, and G. R. Ganger, "Principled workflow-centric tracing of distributed systems", in *Proceedings of the Seventh ACM Symposium on Cloud Computing - SoCC '16*, New York, New York, USA: ACM Press, 2016, pp. 401–414, ISBN: 9781450345255. DOI: 10.1145/2987550.2987568.

[17] OpenTracing, *OpenTracing Data Model Specification.* [Online]. Available: https://github.com/opentracing/specification/blob/master/specification.md (visited on 12/10/2018).

[18] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica, "X-trace: A pervasive network tracing framework", in *Proceedings of the 4th USENIX conference on Networked systems design & implementation (NSDI'07)*, USENIX Association, 2007, p. 20. DOI: 10.1.1.108.2220.

[19] R. R. Sambasivan, R. Fonseca, I. Shafer, and G. R. Ganger, "So, you want to trace your distributed system? Key design insights from years of practical experience", Technical Report, CMU-PDL-14, Tech. Rep., 2014, p. 25.

[20] *The OpenTracing Semantic Specification*, \url{https://github.com/opentracing/specification/blob/maste

[21] *The OpenTracing Semantic Conventions*, \url{https://github.com/opentracing/specification/blob/maste

[22] Cloud Native Computing Foundation, *What is Kubernetes?* [Online]. Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ (visited on 11/29/2018).

[23] OpenStack, *What is OpenStack?* [Online]. Available: https://www.openstack.org/software/ (visited on 11/29/2018).

[24] OpenTracing.io, *What is OpenTracing?* [Online]. Available: https://opentracing.io/docs/overview/what-is-tracing/ (visited on 11/29/2018).

[25] Google LLC, *What is OpenCensus?* [Online]. Available: https://opencensus.io/ (visited on 11/29/2018).

[26] R. Sedgewick and K. Wayne, *Algorithms, 4th Edition - Graphs.* Addison-Wesley Professional, 2011.

[27] D. R. Brillinger, *Time Series: Data Analysis and Theory.* 4. Society for Industrial and Applied Mathematics, 2006, vol. 37, p. 869, ISBN: 0898715016. DOI: 10.2307/2530198. [Online]. Available: https://books.google.pt/books/about/Time%7B%5C_%7DSeries.html?id=PX5HExMKER0C%7B%5C&%7Dredir%7B%5C_%7Desc=y.

[28] H. Liu, S. Shah, and W. Jiang, "On-line outlier detection and data cleaning", *Computers and Chemical Engineering*, vol. 28, no. 9, pp. 1635–1647, 2004, ISSN: 00981354. DOI: 10.1016/j.compchemeng.2004.01.009.

[29] Nikolaj Bomann Mertz, *Anomaly Detection in Google Analytics — A New Kind of Alerting.* [Online]. Available: https://medium.com/the-data-dynasty/anomaly-detection-in-google-analytics-a-new-kind-of-alerting-9c31c13e5237 (visited on 06/06/2019).

[30] JaegerTracing, *Jaeger GitHub*. [Online]. Available: https://github.com/jaegertracing/jaeger (visited on 12/10/2018).

[31] OpenZipkin, *Zipkin Repository*. [Online]. Available: https://github.com/openzipkin/zipkin (visited on 12/10/2018).

[32] R. J. Trudeau and R. J. Trudeau, *Introduction to graph theory*. Dover Pub, 1993, p. 209, ISBN: 0486318664. [Online]. Available: https://books.google.pt/books/about/Introduction%7B%5C_%7Dto%7B%5C_%7DGraph%7B%5C_%7DTheory.html?id=eRLEAgAAQBAJ%7B%5C&%7Dredir%7B%5C_%7Desc=y.

[33] Apache Software Foundation, *Apache Giraph*. [Online]. Available: http://giraph.apache.org/ (visited on 12/03/2018).

[34] J. Shun and G. E. Blelloch, "Ligra: A Lightweight Graph Processing Framework for Shared Memory", Pittsburgh, [Online]. Available: https://www.cs.cmu.edu/%7B~%7Djshun/ligra.pdf.

[35] *NetworkX*, \url{https://networkx.github.io/}.

[36] A. Morin, J. Urban, and P. Sliz, "A Quick Guide to Software Licensing for the Scientist-Programmer", *PLoS Computational Biology*, vol. 8, no. 7, F. Lewitter, Ed., e1002598, Jul. 2012, ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1002598. [Online]. Available: https://dx.plos.org/10.1371/journal.pcbi.1002598.

[37] A. Deshpande, *Surveying the Landscape of Graph Data Management Systems*. [Online]. Available: https://medium.com/@amolumd/graph-data-management-systems-f679b60dd9e0 (visited on 11/24/2018).

[38] J. Celko, "Graph Databases", in *Joe Celko's Complete Guide to NoSQL*, 2013, pp. 27–46, ISBN: 1449356265. DOI: 10.1016/b978-0-12-407192-6.00003-0.

[39] Favio Vázquez, *Graph Databases. What's the Big Deal? – Towards Data Science*, 2019. [Online]. Available: https://towardsdatascience.com/graph-databases-whats-the-big-deal-ec310b1bc0ed (visited on 06/07/2019).

[40] ArangoDB Inc., *ArangoDB Documentation*. [Online]. Available: https://www.arangodb.com/documentation/ (visited on 10/16/2018).

[41] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. Li, M. Marchukov, D. Petrov, L. Puzar, Y. J. Song, and V. Venkataramani, "TAO: Facebook's Distributed Data Store for the Social Graph", [Online]. Available: https://cs.uwaterloo.ca/%7B~%7Dbrecht/courses/854-Emerging-2014/readings/data-store/tao-facebook-distributed-datastore-atc-2013.pdf.

[42] Neo4J Inc., *No Title*. [Online]. Available: https://neo4j.com/docs/ (visited on 10/16/2018).

[43] B. M. Sasaki, J. Chao, and R. Howard, *Graph Databases for Beginners*. 2018, p. 45. [Online]. Available: https://neo4j.com/blog/acid-vs-base-consistency-models-explained/%20https://go.neo4j.com/rs/710-RRC-335/images/Graph%7B%5C_%7DDatabases%7B%5C_%7Dfor%7B%5C_%7DBeginners.pdf?%7B%5C_%7Dga=2.124112970.1994598198.1521285291-1141717847.1521285291%7B%5C&%7D%7B%5C_%7Dgac=1.180373973.1521290471.CjwKCAjw-bLVBRBMEiwAmKSB.

[44] Amenya, *TAO — Facebook's Distributed database for Social Graph*, 2018. [Online]. Available: https://medium.com/coinmonks/tao-facebooks-distributed-database-for-social-graph-c2b45f5346ea (visited on 06/07/2019).

[45] ArangoDB Inc., *ArangoDB Enterprise: SmartGraphs*. [Online]. Available: https://www.arangodb.com/why-arangodb/arangodb-enterprise/arangodb-enterprise-smart-graphs/ (visited on 12/15/2018).

[46] A. Turu, P. Ozge, K. Supervisor, and E. Zimányi, “Université libre de Bruxelles Graph Databases and Neo4J”, Tech. Rep., 2017. [Online]. Available: https://cs.ulb.ac.be/public/%7B%5C_%7Dmedia/teaching/neo4jj%7B%5C_%7D2017.pdf.

[47] K. V. Gundy, *Infographic: Understanding Scalability with Neo4j*. [Online]. Available: https://neo4j.com/blog/neo4j-scalability-infographic/ (visited on 12/15/2018).

[48] T. Dunning and E. Friedman, *Time Series Databases New Ways to Store and Access Data*. 2015, p. 71, ISBN: 9781491917022. [Online]. Available: http://info.mapr.com/rs/mapr/images/Time%7B%5C_%7DSeries%7B%5C_%7DDatabases.pdf%7B%5C%%7D0Ahttp://oreilly.com/catalog/errata.csp.

[49] Tanay Pant, *Ingesting IoT and Sensor Data at Scale – Hacker Noon*, 2019. [Online]. Available: https://hackernoon.com/ingesting-iot-and-sensor-data-at-scale-ee548e0f8b78 (visited on 06/07/2019).

[50] InfluxData, *InfluxDB GitHub*. [Online]. Available: https://github.com/influxdata/influxdb (visited on 12/12/2018).

[51] OpenTSDB, *OpenTSDB*, \url{https://github.com/OpenTSDB/opentsdb}.

[52] C. Churilo, *InfluxDB Markedly Outperforms OpenTSDB in Time Series Data & Metrics Benchmark*. [Online]. Available: https://www.influxdata.com/blog/influxdb-markedly-outperforms-opentsdb-in-time-series-data-metrics-benchmark/ (visited on 12/12/2018).

[53] S. Noor, Z. Naqvi, S. Yfantidou, E. Zimányi, and Z. Zimányi, “Time Series Databases and InfluxDB”, Tech. Rep., 2017. [Online]. Available: https://cs.ulb.ac.be/public/%7B%5C_%7Dmedia/teaching/influxdb%7B%5C_%7D2017.pdf.

[54] S. Jacob, *The Rise of AIOps: How Data, Machine Learning, and AI Will Transform Performance Monitoring*, \url{https://www.appdynamics.com/blog/aiops/aiops-platforms-transform-performance-monitoring/}.

[55] A. Lerner, *AIOps Platforms*, \url{https://blogs.gartner.com/andrew-lerner/2017/08/09/aiops-platforms/}.

[56] S. Nedelkoski, J. Cardoso, and O. Kao, “Anomaly Detection and Classification using Distributed Tracing and Deep Learning”, 2018, [Online]. Available: https://pt.slideshare.net/JorgeCardoso4/mastering-aiops-with-deep-learning.

[57] W. Li, *Anomaly Detection in Zipkin Trace Data*, 2018. [Online]. Available: https://engineering.salesforce.com/anomaly-detection-in-zipkin-trace-data-87c8a2ded8a1.

[58] B. Herr and N. Abbas, *Analyzing distributed trace data*, 2017. [Online]. Available: https://medium.com/@Pinterest%7B%5C_%7DEngineering/analyzing-distributed-trace-data-6aae58919949.

[59] Wikipedia, *Kanban Board*. [Online]. Available: https://en.wikipedia.org/wiki/Kanban%7B%5C_%7Dboard (visited on 12/09/2018).

[60] W. Li, *Anomaly Detection in Zipkin Trace Data*, 2018. [Online]. Available: https://engineering.salesforce.com/anomaly-detection-in-zipkin-trace-data-87c8a2ded8a1 (visited on 12/11/2018).