An Observability Framework for Predicting the Behavior of IoT Systems

Harnidh Kaur

A thesis

submitted in partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND SYSTEMS

University of Washington

2020

Reading Committee:

Eyhab Al-Masri, Chair

Wei Cheng

Sergey Kanzhelev

Program Authorized to Offer Degree:

Computer Science and Systems

University of Washington

Abstract

An Observability Framework for Predicting the Behavior of IoT Systems

Harnidh Kaur

Chair of the Supervisory Committee:
Eyhab Al-Masri
School of Engineering and Technology

Internet of Things (IoT) systems typically consume a number of microservices for completing a business process or task. To this extent, the Quality of Service (QoS) of individual microservices is crucial for measuring the overall performance of the system while identifying potential downtime or sources of failure. By observing the behavior of microservices, it is then possible to identify the root causes of bottlenecks or performance-related issues. In this thesis, we present the Distributed Observability Framework (DOF) for observing the behavior of microservices in a distributed IoT system. Through DOF, it is then possible to identify sources of failure and predict the overall performance of an IoT system more effectively. We have built our DOF observability framework such that it is capable of predicting the overall behavior of IoT systems composed of microservices. In this thesis, we describe the concepts, related work, methodology,

implementation, experimentation, and results. In addition, we present a working prototype that was developed for the purpose of clearly demonstrating the usefulness of this proposed framework. Throughout the thesis, we discuss possible applications and impact of this research work in the development and deployment of IoT systems and how our research work can be incorporated within the current paradigm of distributed tracing.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am highly thankful to many people who were imperative to the completion of this thesis. To start with, I would like to offer my gratitude to my advisor and committee chair Prof. Eyhab Al-Masri who has been incredibly insightful, supportive, and helpful every step of the way in my journey. Our meetings were always fruitful and helped me in comping up with creative solutions and ideas which I implemented in the research and the thesis.

I would also like to express my thanks to the thesis reading committee Dr. Wei Cheng and Sergey Kanzhelev for their valuable feedback and unprecedented support.

Lastly, I would like to thank my friends and family who were always besides me whenever I needed them. They were always present to encourage me and back me with their sincerest wishes and blessings.

Chapter 1
INTRODUCTION

Distributed tracing deals with identifying a path web services undergo or when service requests travel through distributed systems. The telemetry of microservices is the process of capturing the traces and metrics of the microservices in order to measure and analyze the diagnostics of a system. Hence, to determine the behavior of a system involving a number of microservices, distributed tracing plays an increasingly important role in the observability of microservices. Observability can be described as the assessment of the qualities of a system. We formally define observability in Section 3.1 of this thesis. Based on the telemetry results, we establish the relation between the metrics and characteristics of the microservices. This data is then employed to predict the overall behavior of a microservice or how it may influence the microservices' application performance in the future. In an Internet of Things (IoT) system, it is becoming increasingly crucial to keep track of microservices in order to troubleshoot problems associated with latency, to discover services which are causing delays to the overall delivery of service functionality and to determine services that are creating a bottleneck in the system.

Because monitoring and observability are often used interchangeably, we need to identify the basic differences between the two concepts. Monitoring refers to a compilation of metrics, records and logs about a particular system whereas observability determines the distribution of information about a known system. Monitoring helps to predict failures within a deployed system. Observability, on the other hand, helps in understanding the behavior of a system irrespective of a system outage. However, observability does not replace monitoring. In fact, both monitoring and observability complement each other. That is, observability supports achieving an efficient

monitoring of the system. In addition, observability helps in problem solving, planning, development and improving uptime and performance of the system [1]. Through observability, it is then possible to employ decision analysis techniques for making appropriate assessments about the overall behavior of a known distributed software system.

One of the most commonly used Multiple-Criteria Decision Analysis (MCDA) techniques is the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) making it suitable for the analysis of overall behavior of IoT systems. For the purpose of this research work, we employ TOPSIS as a mechanism for generating decisions for the current state of IoT systems primarily based on metrics that can be collected about the IoT system over a period of time through the utilization of distributed tracing. Thus, we propose an observability model which contains a decision-making component that employs TOPSIS in order to generate the decisions for all discovered microservices.

Given that observability helps in achieving an efficient monitoring of an IoT system, with accumulating observability data throughout time, it is then possible to predict the behavior of an IoT system in the future. Hence, another major concept employed in the proposed research work is the ability to predict the behavior of an IoT system using predictions models such as Markov Chains. Markov Chains provides a scientific method comprising of a set of variables that convert from one state to another using defined probabilistic rules [2].Through observability metrics, we can then generate these rules and hence, predict the behavior using Markov Chains. The prediction process involving various system states and utilizing Markov Chains is illustrated in Figure 1.1.

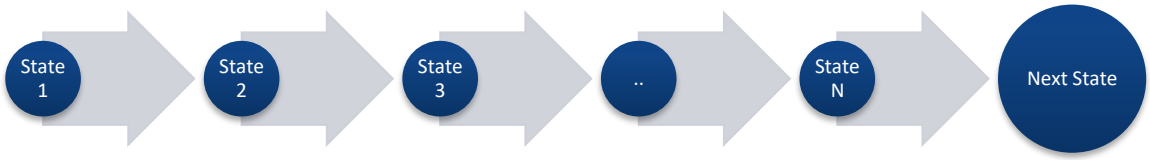

Figure 1.1. Prediction using Markov Chains through Observability.

In recent years, distributed tracing has evolved as effective monitoring tools have the fundamental concurrency and asynchrony of modern software applications [3]. Development of a system that involved distributed tracing for microservices is an intricate task. As the number of processes grow within an application, the concurrency increases as well as the interaction between loosely coupled components that makeup such application. This highlights the importance of tracing the path web services undergo within complex distributed systems. Our proposed research work aims to enhance the architecture of existing distributed tracing capabilities by providing a model to analyze the observability and predict the behavior of microservices. In this manner, we introduce the Distributed Observability Framework (DOF) which is capable of identifying the state of an IoT system at a given point in time using collected metrics that can be captured by telemetry.

Through our proposed DOF framework, it is then possible to evaluate the observability score and make appropriate decisions about the current state of an IoT system. We then use these decisions and observability score to predict the future states of a microservice using Hidden Markov Model. Our DOF framework is capable of measuring the observability score of individual microservice which enable us to identify possible bottlenecks that may occur with the deployment of an IoT system as well as predicting the overall observability of these individual microservices. By determining the individual behavior of microservices, it is then possible to determine the behavior of the overall IoT system that is composed of microservices.

As the number of microservices that deliver similar functionalities increases, the need for an effective mechanism that can identify observability differences that exist among such microservices. By observing the behavior of microservices, it is then possible to determine best microservice combinations that can be utilized in service composition scenarios for building IoT applications. To this extent, as part of our research work, we investigated the following research questions (RQs):

1) Which observability attributes do we need to consider in order to investigate the behavior of a distributed system?
2) How to build a model that efficiently examines and predicts the behavior of microservices?
3) What effect does observability have on the overall performance of an IoT system?

In the upcoming chapters, we demonstrate how our research work through the development of the Distributed Observability Framework (DOF) addresses all of the above research questions. In addition, we provide details on our observability framework, implementation details and validation results which show the effectiveness of our proposed framework.

This thesis focuses chiefly on the observability of microservices in developing and deploying IoT applications in order to enhance their resilience and scalability. Therefore, we do not analyze traditional monolithic web services. Along these lines, it is imperative to differentiate between the microservices and monolithic web service architecture. In a monolithic architecture, the entire application is developed as a single unit containing all of the business logic. However, in a microservices architecture, multiple cooperating software entities that are loosely coupled work together to achieve the goals of a business logic where each microservice focuses primarily on a specific task and it performs that task thoroughly [4]. As a result of this basic distinction, the microservices architecture becomes more easily adaptive, resilient, maintainable, scalable, and

deployable. That is, the microservices' architecture is becoming the most preferred architectural style for developing scalable IoT applications. In addition, the microservices' architecture is more flexible and allows agility while on the other hand, the monolithic architecture is quite the opposite as shown in Figure 1.2.

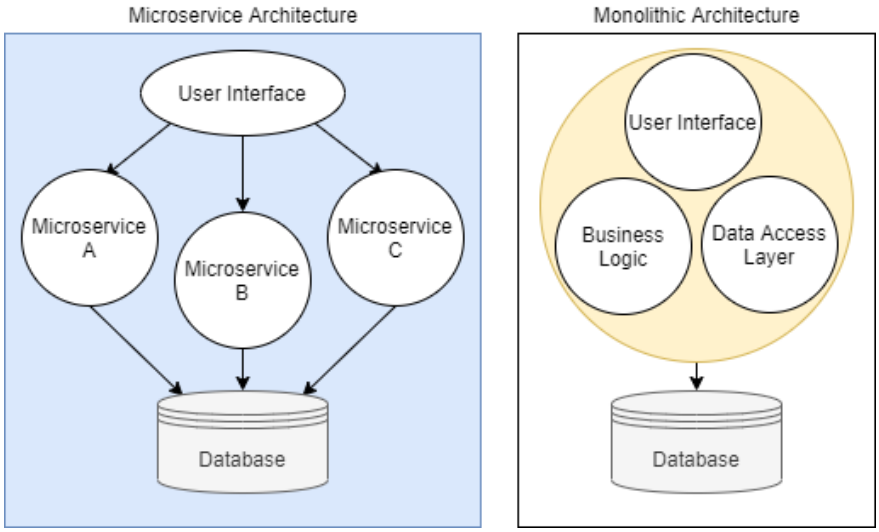

Figure 1.2. Microservice vs Monolithic Architecture.

The microservices' architecture can be composed in multiple different ways or service paths. Therefore, a number of cooperating microservices can sequentially work collectively in a series to deliver a specific functionality or compose a larger task or service. Series service composition refers to the architecture when a sequence of service requests or remote procedure calls (RPCs) is generated one after the other. Alternatively, parallel service composition refers to having multiple service requests or remote procedure calls that begin execution in parallel and the output from all of these services is combined together. In most practical systems, a hybrid composition comes into play which makes use of both the series and parallel compositions [5]. Figure 1.3 presents the various possible microservice composition scenarios.

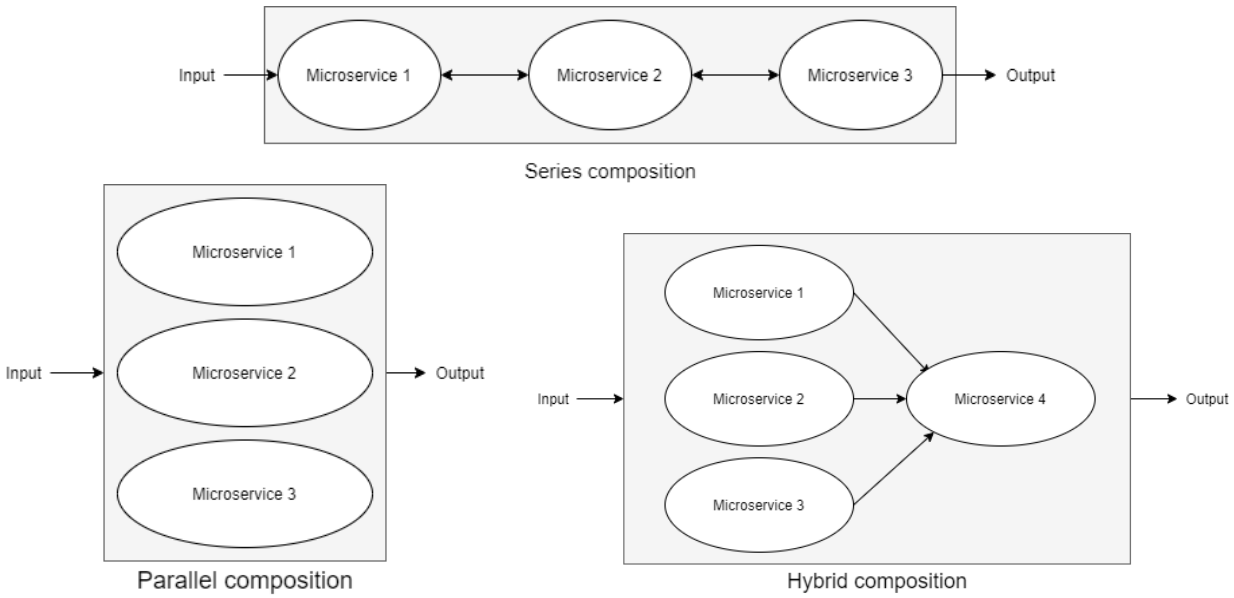

Figure 1.3. (a) Series, (b) Parallel and (c) Hybrid Microservice Composition Scenarios.

Since each subsequent microservice in a series composition path waits for the preceding microservice to complete its operation, series composition can experience significant delays and processing times if one or more microservice introduces an extra processing overhead prolonging the overall service composition response time. In this thesis, we focus our research work primarily on series microservice composition scenarios. Hence, when we are tracing the microservices within an IoT system, observability helps us to identify the bottlenecks that cause or contribute to having high latency, high response times or output errors by examining the composition and analyzing the metrics of all involved microservices. Therefore, we dedicate a major part of our research work throughout this thesis to investigate the effects of observability on the overall performance of distributed software systems.

Chapter 2
RELATED WORK

Distributed tracing enables developers in debugging and monitoring applications in production and facilitates quick resolutions in cases of service errors. In [6], the author provides in depth explanation of the main distributed system concepts that are required for production level engineering. In [7], the authors present Dapper, a large-scale distributed tracing infrastructure. The research work focuses primarily on the lifecycle of remote procedure calls and identifies the concept of trace collection. However, many of the existing research efforts have focused on the usage of distributed tracing and there is little, or no research work conducted in using this information for predicting the behavior of applications composed of microservices. Hence, there very little or no research exploration have been conducted into the observability of microservices over time and how it can help find bottlenecks that may occur or predict the future states of an IoT system.

2.1 Distributed Tracing in Industry

There exists a number of distributed tracing tools in industry with the main objective of helping customers tackle system-related performance issues that may occur and to a good extent provide primitive monitoring capabilities in one way or another. Some of the major industry leaders in this domain include New Relic, Dynatrace, Splunk, among others. While these companies offer some forms of monitoring to their customers through distributed tracing, they vary in the degree to which these monitoring features are offered.

New Relic [8] is one of the most prominent companies in the field of distributed computing. It offers certain features like trace charts, anomalous span detection, among other features are used

to facilitate the error detection in the client application. Similarly, Dynatrace is a software intelligence company with products for the information technology departments and digital business owners of medium and large businesses. The company's services include performance management software for programs running on-premises and on cloud environments [9]. Microsoft Azure has distributed tracing support built-in in its IoT Hub [10]. Nearly all of the existing platforms utilize distributed tracing tools or observability frameworks for maintaining monitoring capabilities and managing software applications.

2.2 DISTRIBUTED TRACING TOOLS

There are a number of distributed tracing tools that exist today, majority of which are open-source. For example, OpenTelemetry [11] is an open-source observability framework that can be integrated into existing software management tools to support distributed tracing capabilities. Zipkin [12] is another popular open-source distributed tracing tool that helps to debug latency issues that may occur as a result of deploying microservices. In addition, Jaeger is a distributed tracing tool that can be used for monitoring and troubleshooting microservices in a distributed system [13]. Furthermore, Splunk emerged as an industry tool for capturing, indexing and maintaining real-time data in a searchable manner with feature-rich front-end interface that includes graphs, reports, dashboards, alerts, among many other features. [14]. Lightstep is a full-context observability platform which can be used to identify bottlenecks and investigate issues in the deployed system [15]. Many of these companies are collaborating to develop interoperable solutions in order to maximize the value benefit that can be achieved through distributed tracing. For example, the Distributed Tracing Working Group emerged based on the need for improving the interoperability between existing distributed tracing tools [16]. Another form of collaboration between industry companies is the addition of New Relic, Dynatrace, Zipkin, Jaeger and Lightstep

to the existing OpenTelemetry SDK [17] [18] [19] [20] [21]. While above-mentioned efforts have been providing helpful capabilities that can lead to the observability of microservices, most of the existing research efforts have failed to focus on observability needs for distributed IoT systems. In such manner, our DOF framework fills this gap and aims to extend existing research efforts to provide an effective methodology measuring the observability and analyzing the overall behavior of the IoT systems over time. Through DOF, it is then possible that IoT systems become more adaptive in cases there exists sources of failure and therefore increases the resilience of IoT systems overall.

2.3 Observability using Distributed Tracing

The observability of microservices is one of the key focus areas of our research work in this thesis. To this extent, we will identify some of the main research efforts that utilized observability in distributed software systems. In [22], the author presents a model for observing and controlling the performance in microservices. The research mainly highlights very basic foundational elements of distributed tracing such as spans, context propagation and the relationship between traces and spans. Investigating this topic further into the domain of microservices monitoring, the research paper [23] introduces a simplified monitoring dashboard with helpful visual insights about microservices. There has been very little or no research that investigated the use of distributed tracing for observing the behavior of distributed software systems. Given that building a model for determining the observability of microservices requires the use of distributed tracing techniques, it is important to gain knowledge about the various visualization and monitoring that also provide details about deployed microservices. The authors in [24] explore the monitoring of several microservices deployed on various cloud distributions. However, the research effort does not provide mechanisms for making decisions related to the overall observability of cloud-based

applications. In addition, the research effort focuses primarily on cloud-based deployment and does not consider edge-based utilization which have become in recent years increasingly essential for the development and deployment of IoT systems. Our proposed DOF framework takes into consideration the essential requirements in the development and deployment of IoT systems which makes it suitable for cloud-based, edge-based and hybrid IoT systems.

2.4 PREDICTION USING DISTRIBUTED TRACING

We identify the behavior of microservices using a decision-making model called TOPSIS. This decision-making technique has been used for analysis of IoT systems in the past [25]. It is a fairly simple but very effective optimization method for making accurate decisions which makes it suitable for generating an overall observability score for the purpose of our research work. As a result of this observability score, it is then possible to determine system's state at any given point in time as well as predict its behavior based on the collection of observability-related metrics.

Results from the decision-making model can then be employed in a prediction model. To this extent, we use the Markov Chain algorithm. A Markov Chain is a scientific system which can be described as a collection of arbitrary variables that convert from one state to another according to defined probabilistic rules [2]. It is a random process with a Markov property. Such random process or what is known as stochastic property provides foundation of a mathematical object which can be defined as a collection of randomly generated variables [2]. Hence, employing Markov chain for our observability framework is very suitable since the behavior of an IoT software system dynamically changes and is based on randomly generated variables that can be collected over time. There are a number of research efforts that have considered the use of Markov Chain for distributed systems.

In [26], the authors presented an extension of Markov Chain which they have generalized for distributed systems to handle the local state and other related factors such as time and concurrency which the called Markov Nets. Additionally, the authors in [27] have employed Markov Chains to analyze the reliability of a distributed system. Some other research efforts have focused on employing Markov Chains for prediction that are application-driven or use-case specific such as the use of Markov Chains for weather forecasting [28], product sales forecasting [29], drought forecasting [30], among others.

Other research efforts have applied Markov Chains to distributed systems. In [31], the authors developed a model for using Markov Chains to monitor complex distributed systems which utilized the discrete time Markov Chains. The research efforts in [22] are very relevant to our proposed observability framework since we are evaluating the observability within distributed IoT systems with the usage of Markov Chains. However, existing research efforts failed to consider the essential elements that compose an IoT systems such as IoT devices, device capabilities, deployment scenarios (e.g., edge-based versus cloud-based or hybrid deployment), service delivery, quality of service (QoS) factors, among many others. Our proposed observability framework attempt to fill this gap by considering essential IoT system features for the purpose of predicting the overall observability of IoT systems over time using Markov Chains. We introduce our observability framework in Chapter 3.

Chapter 3

DISTRIBUTED OBSERVABILITY FRAMEWORK (DOF)

In order to build our Distributed Observability Framework (DOF), it is essential to consider the basic entities and elements that will be used throughout the implementation of this framework. Hence, we need to first define the terms of such entities or elements that will employed throughout this research work. We have carried out this research by developing a system for observability and we shall provide the technical details of the methodology that we followed in this chapter. With this framework, we can perform the following tasks for IoT systems:

1. Measure the observability of IoT systems: The decisions for each microservice are made which lead to the computation of the observability score based on historical metrics.

2. Identify bottlenecks that cause performance issues in an IoT system: We shall look at how the framework aids in taking decisions related to improvement in performance of an IoT system.

3. Predict the behavior of microservices and the overall IoT system: With our prediction model, which is built using Markov Chains, we can predict the observability of individual microservices and hence, calculate the predicted observability of the entire IoT system.

3.1 DEFINITIONS

We define some of the commonly used terms or entities throughout this research work which are essential components of our proposed DOF framework. We first defined observability as follows.

3.1.1 Observability

Observability is defined as the ability of a system to produce adequate data or statistics such that it can be monitored over a duration of time.

3.1.2 Distributed observability

Distributed observability is defined as the ability of a distributed system whose resources are distributed throughout a network of resources to produce adequate data or statistics such that it can be monitored over a duration of time.

3.1.3 Distributed Observability Framework (DOF)

Distributed Observability Framework (DOF) is defined as a framework consisting of distributed tracing elements that is designed to measure the (a) observability, (b) identify bottlenecks and (c) predict the behavior of microservices in distributed IoT systems.

3.1.4 Observability Score

Observability Score is defined as the measure of the degree to which the system is capable of being observed adequately to produce data or statistics such that it can be monitored over a period of time. We measure observability score at the following levels by using DOF: (a) Trace level, (b) Microservice-level and (c) System-level. Further details about the calculation at different levels are in the Chapter 4.

3.2 HYPOTHESIS

H1: If we measure the observability of microservices, we can identify factors that may impact or influence the service level offering by an Internet of Things (IoT) system.

Throughout this research work, we have statistical data of what extent does the performance and observability of individual microservices have on the overall performance and observability of the IoT system as a whole. We assume that the observability of an IoT system is primarily dependent

on the observability score of individual microservices that compose such IoT system. We assume that such individual microservices are designed to deliver a specific functionality and that quality of services (QoS) metrics such as response time, throughout, availability, among others are measurable. To this extent, we designed and developed a working prototype in order to measure the observability scores of individual microservices which will help in holistically evaluating and predicting the overall behavior of an IoT system as a whole. The prototype aims to show the effectiveness of our proposed Distributed Observability Framework (DOF), run sufficient tests to test the formulated hypothesis (H1) and demonstrate the overall accuracy of our proposed prediction model in making decisions.

3.3 System Architecture

In this section, we discuss the architecture of our proposed Distributed Observability Framework (DOF). The goals of the DOF framework is to support the overall observability of the behavior of IoT systems which can be accomplished using a number of tasks. First, we need to measure the overall observability of an IoT systems. However, to measure the overall observability, we need to determine the individual observability metric for every microservice that exists or consumed within the IoT system. As a result, the decisions for each microservice are made which primarily evaluates into computing the observability score for individual microservices based on historical metrics. In this section, we will discuss the architecture of our Distributed Observability Framework (DOF).

In addition, as part of the DOF framework, we then can identify any system degrades or bottlenecks that may cause performance issues that occur within an IoT system. In this approach, we investigate how the framework can be employed in formulating decisions that pertain to improving the overall performance of an IoT system. Furthermore, we utilize the DOF framework

to predict the behavior of individual microservices that compose an IoT system. Through our proposed prediction model, which is built using Markov Chains, it is then possible to predict the observability of individual microservices and hence, calculate the predicted observability of the entire IoT system as a whole or holistically. For the purpose of this research work, we will test the hypothesis based on a real-world dataset (called WS-Dream dataset [32]) to be used as the metrics needed for our observability framework to perform computations. Figure 3.1 provides an overview of the proposed Distributed Observability Framework (DOF) system architecture.

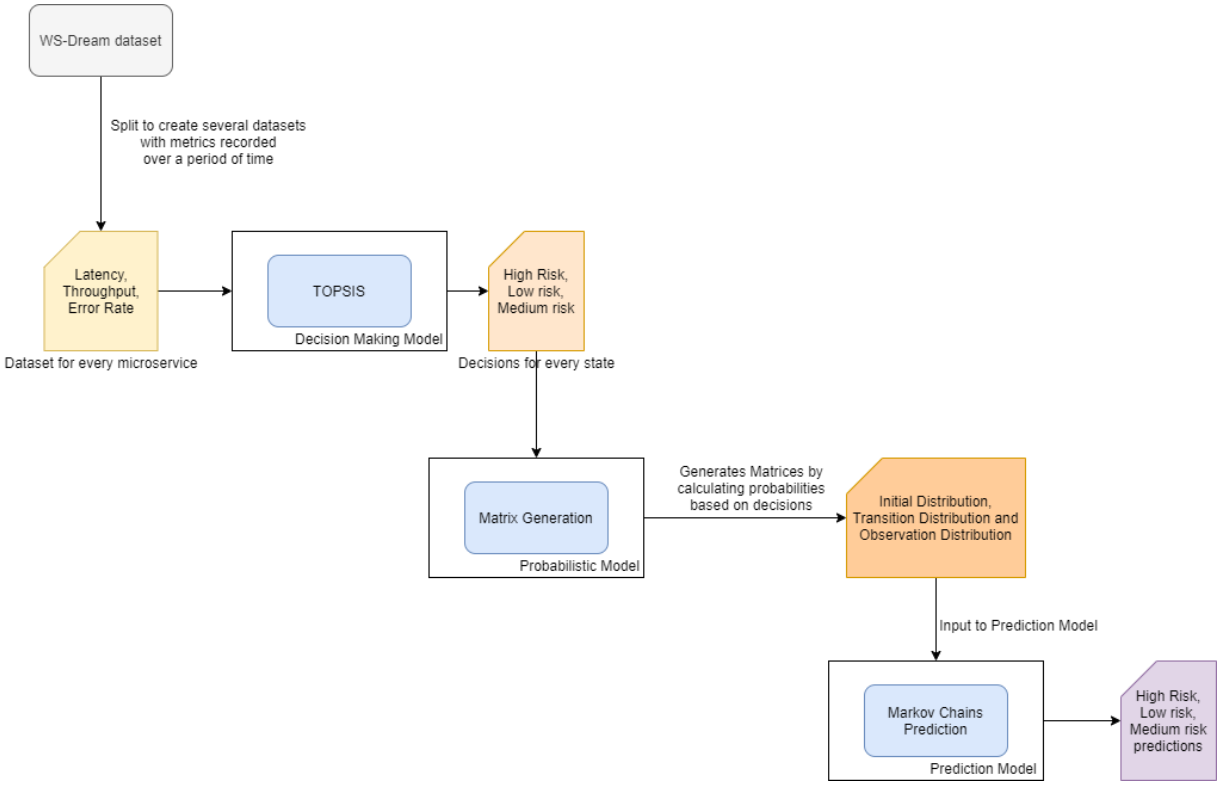

Figure 3.1. System Architecture of Distributed Observability Framework (DOF).

As presented in Figure 3.1, the DOF framework contains multiple components to achieve its tasks. We describe these components in more details in the following sub-sections.

3.3.1 Decision-Making Model

Based on observability analysis and computations conducted by the DOF framework, it is then possible to categorize the risk associated with consuming a discovered individual microservice within a defined IoT system into one of three main risk categories: (a) high risk, (b) moderate risk and (c) low risk. The DOF framework uses can utilize a classification algorithm for this identification. Given that there exists multiple criteria for making a decision about the state of existing microservices, the DOF framework employs the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) which enables the computation of an observability score for each microservices and hence allows us to establish a clear idea about the state of every microservice consumed within an IoT system. Using such observability scores over a duration of time, we can then predict the next behavioral state in the future. Figure 3.2 presents the steps involved in the decision-making process of the DOF framework. We discussed the implementation details of the decision-making model in Section 4.1.2.

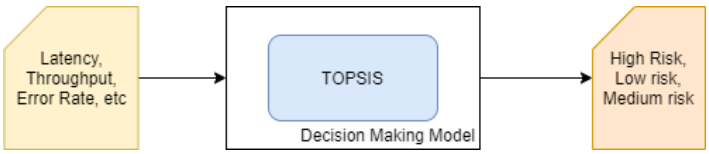

Figure 3.2. Decision-Making Model.

3.3.2 Probabilistic Model

The probabilistic model is used to generate the probability distributions which are essential to the overall determination of the observability of the IoT system. These distributions are required by the Markov Chains algorithm and must be created from the output of the Decision-Making Model. Such distributions form the basis of the prediction model which are carried out by the Prediction Model component as shown in Figure 3.3.

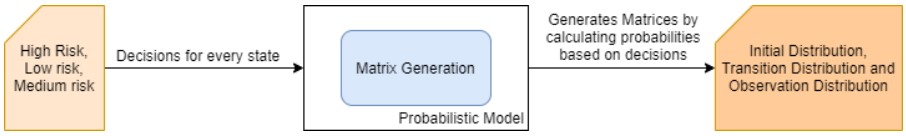

Figure 3.3. Probabilistic Model.

3.3.3 Prediction Model

We employ the Markov Chain algorithm to analyze the behavior of microservices. Based on the collected data or metrics, DOF uses use a Markov Chain implementation to predict the future states of the IoT system. However, this requires a suitable amount of transformation on the existing data to calculate various probabilities attached with the state of microservices. We use the Hidden Markov Model by Tensorflow for our prediction model. We use randomly generated data for the initial implementation of the prediction model. Throughout the prediction model, the DOF framework is capable of predicting the consecutive states based upon historical performance scores (*which is defined in the upcoming sections*). This score is obtained for each service request for an API call by using the decision-making model and the traces obtained from the IoT system through distributed tracing tools as shown in Figure 3.4.

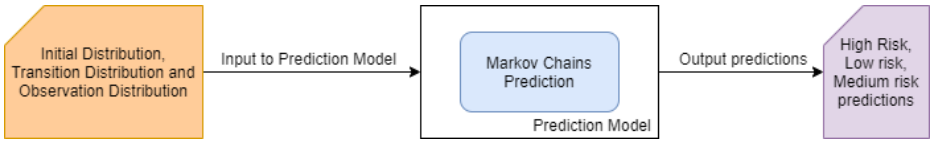

Figure 3.4. Prediction Model.

Chapter 4
EXPERIMENTATION AND EVALUATION

This chapter describes the details of the experimental setup and the different types of tests that we conducted using variations in our input and by generating several service compositions to evaluate the DOF that we developed. This process is crucial in our research since this reveals the performance assessment as well as demonstrate the practicality of our framework.

4.1 EXPERIMENTATION SETUP

The empirical analysis is performed using a quad-core PC with Windows 10 Version 1909 OS Build 18363.1198 operating system, 16GB of memory and CPU speed of 2.13 GHz. The average runtimes, CPU Usage and Memory Usage of the experiments are shown in Table 4.1.

Table 4.1. Experiment Runtimes

Component	Total Runtime of 5786 microservices each with 330 traces	Average Runtime per microservice with 330 traces	CPU Usage (approx.)	Memory Usage (approx.)
Decision Making	9 seconds, 302 milliseconds	1.6 milliseconds	19.6%	1311 MB
Probability Matrix Generation	8 seconds, 673 milliseconds	1.49 milliseconds	18.1%	1524 MB
Prediction	3 minutes, 39 seconds	378 milliseconds	11.5%	134 MB

4.1.1 Dataset

The WS-Dream is a publicly available dataset that contains the real-world Quality of Service (QoS) metrics of 5825 web services. This dataset comprises of the recorded QoS attributes including response time and throughput metrics that have been collected from 339 consumers of these web services. The response time is measured in seconds and throughput is measured in kbps. The total number of traces that we used is 1,922,250. Using this real-world data, we have

orchestrated a sample IoT system for the simulation of how observability can be computed, visualized, and predicted for an IoT system. We assume that these metrics in the dataset represent those of microservices that can be used to compose an IoT system. In addition, we assume that the traces collected represent observing the behavior of microservices over a duration of time. We also assume that the traces are generated from low-power IoT devices with IoT systems that have uniformity in signals and behavior. We use this information to fine-tune our decision-making model such that it can identify an observability score for each discovered microservice. Given a path that consists of a number of microservices, we can then determine the overall observability for each path. For all discovered paths in a defined IoT system, we are able to then identify the observability for the entire system.

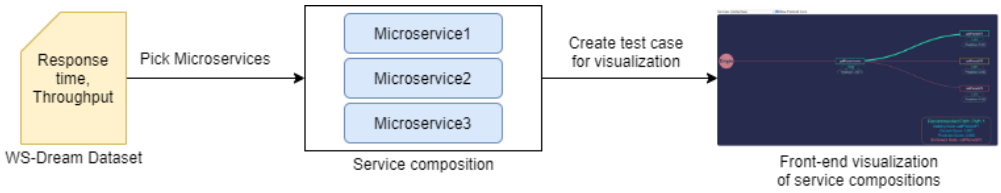

Figure 4.1. Creating service composition from dataset.

4.1.2 Decision-Making Model

The WS-Dream dataset contains metrics of two QoS attributes: (a) response time in seconds and (b) throughput in kbps for a total number of 5825 web services. We use this data as input to our Distributed Observability Framework (DOF). Since the employed dataset has two QoS attributes (response time and throughput), we applied a two-dimensional decision-making process. However, the implementation of decision-making model is flexible such that it can easily support additional dimensions or dynamic variables depending on the input matrix which contains the metrics. We use the metrics from the WS-Dream dataset to generate the output, and then we

separate the microservices into three categories representing the risk associated with integrating a microservice in a discovered path or service composition scenario. These categories include:

1. High risk: A microservice that is designated as high risk is one that poses a critical performance degradation to a defined system in cases it crashes or continues its normal operation. In the event of a failure, it would be of the utmost priority to resolve the issues associated with a high-risk microservice. Quick fixes would be needed in cases such a microservice malfunctions during software production stage since it would cause the entire system to be hampered.
2. Medium risk: A microservice that is designated as medium risk is one that can cause some parts or portion of the system to degrade its performance noticeably but this microservice does not cause complete malfunctioning of the entire system.
3. Low risk: A microservice that is designated as low risk is one that poses minimal degradation or performance deterioration to the system in cases it crashes or malfunctions. Such microservices are usually associated with non-critical features of the system.

The decision-making model has been implemented using Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS). The model is scalable to accommodate any number of attributes considered as inputs for generating a decision-making output. Since, we have employ the WS-Dream dataset, we only have two attributes as discussed earlier (i.e., response time and throughput). From the dataset, we retrieve the traces in CSV format. Then, these traces are divided into two segments. The first segment contains all the response time values whereas the second segment contains all the corresponding throughput values. Hence, we transform the values into two segments and generate an observability matrix that serves as an input to the TOPSIS model. Hence, we compute our input matrix, M, as follows:

$$M = \begin{pmatrix} \text{ResponseTime}_1 & \text{Throughput}_1 \\ \text{ResponseTime}_2 & \text{Throughput}_2 \\ \text{ResponseTime}_3 & \text{Throughput}_3 \\ \cdots & \cdots \\ \cdots & \cdots \\ \text{ResponseTime}_n & \text{Throughput}_n \end{pmatrix}$$

The next input to the TOPSIS is the weight distribution matrix or W represents the weights (between 0 and 1) associated with the attribute's response time and throughput. The weights represent the priority associated with each QoS dimension. The higher the weight translates into higher priority and therefore more emphasis will be associated with that attribute. For example, if the response time and throughput are equal contributors to the overall state of a microservice, the weight distribution matrix, W, will be:

$$W = \begin{pmatrix} 0.5 & 0.5 \end{pmatrix}$$

The third input to the TOPSIS model is the indication about whether the criteria is beneficial to the performance or not. That is, minimizing the value of a particular attribute means better performance whereas maximizing the value of another attribute could mean better performance. For example, lower response time would be desired and therefore resulting in better performance of the microservice's behavior whereas higher response time represents deteriorating or poor performance of a microservices. Therefore, the decision-making model should minimize such values for this type of attribute. Therefore, our indication matrix, I, can be defined as:

$$I = \begin{pmatrix} \text{unfavorable} & \text{favorable} \end{pmatrix}$$

where bad represents the attribute that contributes to low performance with higher value (higher response time is unfavorable) and good represents the attribute which gives better performance with higher values (higher throughput is favorable).

Based on these three generated matrices, our TOPSIS implementation evaluates the Observability Score for every state of each microservice totaling to 339 observability score for each microservice in the WS-Dream dataset. We normalize the values of the QoS metrics and therefore the resulting observability score range between 0 and 1, respectively. We use this observability score to categorize the microservices based on the risk as follows:

1. High Risk – Observability Score is lower than 0.33
2. Medium Risk – Observability Score equals 0.33 and less than or equal to 0.66
3. Low Risk – Observability Score is greater than 0.66

Hence, our model finally tags the associated category corresponding to each trace of the microservices. A sample output for some traces in the dataset is as follows:

Table 4.2. Sample Illustration of Observability Score Categorization

Observability Score	**Decision**
0.467296	mediumRisk
0.720477	lowRisk
0.023428	highRisk

Based on the categorization of the various microservice's states, we calculate the average observability of a microservice by computing the average of observability scores of all traces for a particular microservice. The model is implemented in Node.js.

4.1.3 Probabilistic Model

The Probability Matrix Generation or Probabilistic Model is constructed as the bridge between decision-making model and the prediction model. The decision-making model provides DOF with the decision for the state of each microservice based on which we then generate the overall observability score of the respective microservices. However, the output of this model cannot be

consumed directly by the prediction model. Since we are using Markov Chain prediction, the prediction model requires the input as three probability distributions:

4.1.3.1 **Initial Distribution**

Initial Distribution is the probability for the starting value of the system for the prediction. It refers to the probability of the initial state to fall in a category, i.e., low-risk, medium-risk or high-risk. We calculate initial distribution by calculating the probability of the state in one microservice to be low-risk, medium-risk or high-risk. The formulae used are as follows:

$$\text{Probability of low} - \text{risk state} = \frac{\text{Number of traces in low}-\text{risk state}}{\text{N}} \quad (4.1)$$

$$\text{Probability of medium} - \text{risk state} = \frac{\text{Number of traces in medium}-\text{risk state}}{\text{N}} \quad (4.2)$$

$$\text{Probability of high} - \text{risk state} = \frac{\text{Number of traces in high}-\text{risk state}}{\text{N}} \quad (4.3)$$

where N represents the total number of traces that can be collected for a discovered microservice. Using the three probabilities generate with the above formulae, we get initial distribution matrix:

$$\text{Initial Distribution} = \begin{pmatrix} \text{Low-risk probability} & \text{Medium-risk probability} & \text{High-risk probability} \end{pmatrix}$$

4.1.3.2 **Transition Distribution**

The Transition Distribution refers to the probability for one state to be followed by the next state (i.e., low-risk state to be followed by low-risk state, medium-risk state and high-risk state). Similarly, the probability of medium-risk state to be followed by low-risk state, medium-risk state and high-risk state and then finally, the probability of high-risk state to be followed by low-risk state, medium-risk state and high-risk state. For the computation of the transition matrix, we assume that the traces for a microservice are ordered, non-overlapping and equally distributed over time. The calculation is performed using the following formulae:

$$\text{Probability of low} - \text{risk state to be followed by low} - \text{risk state, } \mathbf{pLL} = \frac{\text{nLL}}{\text{N}} \quad (4.4)$$

$$\text{Probability of low} - \text{risk state to be followed by medium} - \text{risk state, } \mathbf{pLM} = \frac{\text{nLM}}{\text{N}} \quad (4.5)$$

$$\text{Probability of low} - \text{risk state to be followed by high} - \text{risk state, } \mathbf{pLH} = \frac{\text{nLH}}{\text{N}} \quad (4.6)$$

$$\text{Probability of medium} - \text{risk state to be followed by low} - \text{risk state, } \mathbf{pML} = \frac{\text{nML}}{\text{N}} \quad (4.7)$$

$$\text{Probability of medium} - \text{risk state to be followed by medium} - \text{risk state, } \mathbf{pMM} = \frac{\text{nMM}}{\text{N}} \quad (4.8)$$

$$\text{Probability of medium} - \text{risk state to be followed by high} - \text{risk state, } \mathbf{pMH} = \frac{\text{nMH}}{\text{N}} \quad (4.9)$$

$$\text{Probability of high} - \text{risk state to be followed by low} - \text{risk state, } \mathbf{pHL} = \frac{\text{nHL}}{\text{N}} \quad (4.10)$$

$$\text{Probability of high} - \text{risk state to be followed by medium} - \text{risk state, } \mathbf{pHM} = \frac{\text{nHM}}{\text{N}} \quad (4.11)$$

$$\text{Probability of high} - \text{risk state to be followed by high} - \text{risk state, } \mathbf{pHH} = \frac{\text{nHH}}{\text{N}} \quad (4.12)$$

where

nLL = Number of traces where low-risk state is followed by low-risk state,

nLM = Number of traces where low-risk state is followed by medium-risk state,

nLH = Number of traces where low-risk state is followed by high-risk state,

nML = Number of traces where medium-risk state is followed by low-risk state,

nMM = Number of traces where medium-risk state is followed by medium-risk state,

nMH = Number of traces where medium-risk state is followed by high-risk state,

nHL = Number of traces where high-risk state is followed by low-risk state,

nHM = Number of traces where high-risk state is followed by medium-risk state,

nHH = Number of traces where high-risk state is followed by high-risk state, and

N is the total number of traces in that microservice.

Using the nine probabilities generated with the above formulae, we get transition distribution as:

$$\text{Transition Distribution} = \begin{pmatrix} \text{pLL} & \text{pLM} & \text{pLH} \\ \text{pML} & \text{pMM} & \text{pMH} \\ \text{pHL} & \text{pHM} & \text{pHH} \end{pmatrix}$$

4.1.3.3 **Observation Distribution**

Observation Distribution (OD) contains the following distributions:

1. mean value of the observability score in low-risk state, medium-risk state and high-risk state for the microservice

2. standard deviation of the observability score in low-risk state, medium-risk state and high-risk state for the microservice

The mean values are calculated by using the following formulae:

$$\text{Mean observability score in low-risk state, mL} = \frac{\sum \text{observability Score in low-risk state}}{\text{nL}} \quad (4.13)$$

$$\text{Mean observability score in medium-risk state, mM} = \frac{\sum \text{observability Score in medium-risk state}}{\text{nM}} \quad (4.14)$$

$$\text{Mean observability score in high-risk state, mH} = \frac{\sum \text{observability Score in high-risk state}}{\text{nH}} \quad (4.15)$$

where

nL = number of traces in low-risk state,

nM = number of traces in medium-risk state, and

nH = number of traces in high-risk state.

In a similar context, the standard deviations from observability score for low-risk states, medium-risk states and high-risk states are calculated by using the following formulae:

$$\text{Standard deviation in low-risk state, sL} = \sqrt{\frac{\sum(\text{mL} - \text{observability Score in low-risk state})^2}{\text{nL}}} \quad (4.16)$$

$$\text{Standard deviation in medium-risk state, sM} = \sqrt{\frac{\sum(\text{mM} - \text{observabilityScore in med-risk state})^2}{\text{nL}}} \quad (4.17)$$

$$\text{Standard deviation in high-risk state, sH} = \sqrt{\frac{\sum(\text{mH} - \text{observability Score in high-risk state})^2}{\text{nL}}} \quad (4.18)$$

where

mL = mean observability score in low-risk state,

mM = mean observability score in medium-risk state,

mH = mean observability score in high-risk state,

nL = number of traces in low-risk state,

nM = number of traces in medium-risk state, and

nH = number of traces in high-risk state.

Finally, the observation distribution is obtained which is shown as follows:

$$\text{Observation Distribution} = \begin{pmatrix} mL & mM & mH \\ sL & sM & sH \end{pmatrix}$$

The three probability distributions that are generated serve as the input to the DOF prediction model. The probabilistic model is developed in Node.js.

4.1.4 Prediction Model

The Prediction Model is built by using the concept of Hidden Markov Chains. It consumes the output probability distributions generated by the Probabilistic Model in Section 4.1.3 and generates the output containing the prediction scores for every microservice discovered within the dataset. The predicted observability score is generated at the microservice-level.

We implemented this model using the Hidden Markov Model in TensorFlow (more details in Experimentation Setup section). In employing this prediction algorithm, it is assumed that the probability of next state depends on the previous state attained from earlier events. Hence, in our case, the probability of the next state of a microservice to be categorized as low-risk, medium-risk or high-risk state depend primarily on the earlier states attained by the microservice (e.g., low-risk, medium-risk or high-risk).

In our implementation, the Prediction Model generates the predicted observability score based on initial distribution, transition distribution and observation distribution and generates a result matrix shown in tabular format in Table 4.3.

Table 4.3. Predicted Results from our Observability Model

Microservice ID (0-5824)	**Predicted Observability**
0	Predicted Observability Score 0
1	Predicted Observability Score 1
2	Predicted Observability Score 2
…	…
…	…
…	…
5824	Predicted Observability Score 5824

The output is generated for the 5825 microservices discovered in the WS-Dream dataset. The index starts at 0 and ends at 5824. We describe more about the different experimentation done in the following sections. The prediction model is coded using Python and the other technologies used are Tensorflow and Jupyter Notebook.

4.2 EXPERIMENTATION RESULTS

We first analyze the results generated by the different components in our framework in this section.

4.2.1 Decision-Making Model

The decision-making model takes the two segments from WS-Dream dataset as the input and generates the output as decisions and observability scores for each trace of each discovered microservice. This data is then stored in a CSV file corresponding to every microservice. Table 4.4 presents a snippet of output for one of the microservices (ID: 0):

Table 4.4. Snippet from the Output of DOF's Decision-Making Model

Index of Trace	Data (Response Time, Throughput)	Observability Score	Decision	Response Time	Throughput
134	1.824;1.096	0.688884	lowRisk	1.824	1.096
134	3.132;0.638	0.626518	mediumRisk	3.132	0.638
135	9.312;0.214	0.367893	mediumRisk	9.312	0.214
136	6.740;0.296	0.480071	mediumRisk	6.74	0.296
137	15.293;0.130	0.114482	highRisk	15.293	0.13
139	1.322;1.512	0.726152	lowRisk	1.322	1.512
139	6.274;0.318	0.4996	mediumRisk	6.274	0.318
140	5.309;0.376	0.539198	mediumRisk	5.309	0.376
141	9.743;0.205	0.348549	mediumRisk	9.743	0.205
142	12.332;0.162	0.231505	highRisk	12.332	0.162

The first column is the index representing the trace for which the output is generated. The second column represents the raw CSV data used to make decisions for the trace. The third column represents the observability score corresponding to the trace index. The fourth column is the decision (low-risk, medium-risk or high-risk) taken based on observability score. The fifth and sixth columns represent the response time and throughput values respectively. The last two columns are mainly used to compute the averages for the overall microservice.

4.2.2 Probabilistic Model

The probabilistic model generates the three distributions: (a) initial distribution, (b) transition distribution and (c) observation distribution. Below is a sample output for one of the microservices (ID: 0).

$$\text{Initial Distribution} = \begin{pmatrix} 0.781 & 0.206 & 0.012 \end{pmatrix}$$

$$\text{Transition Distribution} = \begin{pmatrix} 0.64 & 0.14 & 0.01 \\ 0.13 & 0.07 & 0.01 \\ 0.01 & 0.00 & 0.00 \end{pmatrix}$$

$$\text{Observation Distribution} = \begin{pmatrix} 0.755 & 0.544 & 0.162 \\ 0.050 & 0.078 & 0.059 \end{pmatrix}$$

4.2.3 Prediction Model

The prediction model retrieves all segments generated by the probabilistic model as input and generates a matrix comprising of: (a) microservice ID and (b) the corresponding predicted observability. Then, we store the output matrix in a CSV file. Table 4.5 presents a snippet from the output of prediction model:

Table 4.5. Snippet from the Output of DOF's Prediction Model

Microservice ID (0-5824)	**Predicted Observability**
1000	0.448072824
1001	0.741162300
1002	0.578498864
1003	0.525054232
1004	0.584706012
1005	0.454344799
1006	0.569478292
1007	0.38947282
1008	0.557292167
1009	0.596661015

4.3 EVALUATION

Since we have a dataset with the quality of service metrics for a volume of 1,922,250 traces, we have analyzed random subsets from the results to evaluate the results being generated by the overall framework. We perform a series of use cases for the purpose of analyzing the generated data and testing the hypothesis H1. For the decision-making model, we generated the observability scores by varying the priority associated with the weights of response time and throughput. We describe the results of our evaluation of these test cases in the following sub-sections.

4.3.1 ResponseTime-Driven Use Case

In this use case, the priority is associated primarily or entirely on response time which means that only the response time values would influence the decisions determined by our decision-making model. Throughput has zero weightage indicating that it is not a priority. This test case represents fog-based mission critical scenario of an IoT system scenarios involving fog-based environments where local network bandwidth is high but the processing of microservice requests need to be as quick as possible. Table 4.6 presents the results generated by our decision-making model for one of the microservices which we reference as “microservice 1”.

As shown in Table 4.6, the trace having the highest response time of 10.118 seconds is associated has the lowest observability score of 0.093888. Furthermore, the trace of the microservice having the lowest response time of 0.798 seconds has the highest observability score of 0.853464. In this use case, we wish to minimize the response time representing better performance since the priority is entirely to 1 for this attribute. As shown in Table 4.6, as the response time increases, the observability score decreases reflecting the differences in the microservice trace. Furthermore, for microservice 1, traces which have similar response times also share similar observability scores (as italicized in the table).

Table 4.6. Traces' Cross-section for Microservice 1

Response Time	**Throughput**	**Observability Score**	**Decision**
0.820	10.975	0.851671	lowRisk
0.977	9.211	0.838875	lowRisk
0.798 (12.68 times better)	11.278	***0.853464 (highest)***	***lowRisk***
0.900	10.000	0.845151	lowRisk
10.118 (12.68 times worse)	0.889	***0.093888 (lowest)***	***highRisk***
4.325	2.080	0.566015	mediumRisk
1.405	6.405	0.803993	lowRisk
0.883	10.192	0.846536	lowRisk
0.990	9.090	0.837816	lowRisk
4.967	1.811	0.513692	mediumRisk
7.949	1.132	0.27066	highRisk
3.446	2.611	0.637653	mediumRisk
2.713	3.317	0.697392	lowRisk
0.849	10.600	0.849307	lowRisk
0.963	9.345	*0.840016 (similar score)*	*lowRisk*
0.957	9.404	*0.840505 (similar score)*	*lowRisk*
6.343	1.418	0.401548	mediumRisk

4.3.2 Throughput-Driven Use Case

In this use case, the priority is set for throughput which indicates more emphasis is associated with this variable compared to that of response time. This use case represent IoT system scenario that involve heavy data transfers and require sufficient bandwidth. In such cases, we need to identify microservices that possess high throughput. Table 4.7 presents the results generated by our decision-making model for one of the microservices which we reference as "microservice 2".

Table 4.7. Traces' Cross-section for Microservice 2

Response Time	**Throughput**	**Observability Score**	**Decision**
0.199	20.100	0.068353	highRisk
0.183	*21.857*	*0.074045 (similar score)*	*highRisk*
0.182	*21.978*	*0.074437 (similar score)*	*highRisk*
0.184	***21.739 (10.22 times worse)***	***0.073662 (lowest)***	***highRisk***
0.037	108.108	0.353453	mediumRisk
0.038	105.263	0.344236	mediumRisk
0.04	100.000	0.327187	highRisk
0.173	23.122	0.078139	highRisk
0.173	23.121	0.078139	highRisk
0.069	57.971	0.191035	highRisk
0.019	210.526	0.685233	lowRisk
0.020	200.000	0.651134	mediumRisk
0.034	117.640	0.384354	mediumRisk
0.034	117.647	0.384354	mediumRisk
0.021	190.476	0.620282	mediumRisk
0.018	***222.222 (10.22 times better)***	***0.723122 (highest)***	***lowRisk***
0.159	25.157	0.084735	highRisk

As can be seen from Table 4.7, the microservice 2 trace that is associated with the highest observability score of 0.723122 represents the microservice trace having the highest throughout of 222.222 kbps. Similarly, the microservice 2 trace that possess the lowest observability score of 0.073662 reflects that of a microservice trace having the lowest throughput of 21.739 kbps. As the throughput increases, the observability score also increase. Unlike response time, we wish to identify those traces having higher throughput since higher throughput values indicate higher bandwidth. Furthermore, for microservice 2, traces which have similar response times also share similar observability scores (as italicized in the table).

4.3.3 Generic Use Case

In this use case, both response time and throughput share equal weights or priority distribution (0.5, 0.5). The degree of variation among response time and throughput values across traces will be treated of equal importance, respectively. Table 4.8 presents the results generated by our decision-making model for one of the microservices which we reference as "microservice 3".

Table 4.8. Traces' Cross-section of Microservice 3

Response Time	**Throughput**	**Observability Score**	**Decision**
1.923	*1.040*	*0.683087192 (similar score)*	*lowRisk*
0.996	2.008	0.765089922	lowRisk
1.824	*1.096*	*0.688883628 (similar score)*	*lowRisk*
3.132	0.638	0.626517674	mediumRisk
9.312	0.214	0.367893353	mediumRisk
6.740	0.296	0.480070877	mediumRisk
15.293 (20.42 times worse)	***0.13 (20.54 times worse)***	***0.11448155 (lowest)***	***highRisk***
1.322	1.512	0.726152365	lowRisk
6.274	0.318	0.499600071	mediumRisk
5.309	0.376	0.53919779	mediumRisk
9.743	0.205	0.348548797	mediumRisk
12.332	0.162	0.231504638	highRisk
0.803	2.490	0.801410688	lowRisk
0.823	2.430	0.796916937	lowRisk
1.344	1.488	0.724155456	lowRisk
0.749 (20.42 times better)	***2.67 (20.54 times better)***	***0.814840518 (highest)***	***lowRisk***
6.021	0.332	0.51011452	mediumRisk

As can be seen from Table 4.8, the microservice 3 traces that possesses the highest observability score of 0.81484 has a moderately sufficient throughput of 2.67 kbps and low response time of 0.749 seconds. These values represent 20.42 and 20.54 times better than the worst traces having 15.293 seconds for response time and 0.13 kbps for throughput, respectively. This

is an indicator that the observability score can help identify factors that may influence the performance of the overall IoT system. For example, assume that the microservice 3 trace having lowest observability score of 0.11448 is employed in the IoT system. Through this observability score, it is then possible to identify the root causes for this lower score which can be seen from Table 4.8 as having 15.293 seconds for response time and 0.13 kbps for throughput, respectively. Therefore, observability helps identify the behavior of the microservice which proves the hypothesis H1 suggesting that, given observability scores, we are able to identify the causes that may influence the overall behavior of the IoT system. The use cases presented in this section all show that the observability score supports the hypothesis and that having deteriorations in response time or throughput are likely to have significant impact on the overall performance of the microservice's behavior (20.54 times worse), given in the Table 4.8, we can clearly see that for the trace with the best combination of response time and throughput, the observability score is the highest and for the trace with the worst combination of metrics, the observability score is the lowest in this cross-section. Furthermore, for microservice 3, traces which have similar response times also share similar observability scores (as italicized in the table).

Therefore, we can conclude that the decisions identified by our distributed observability accurately reflect the state of the microservice. To validate these result and support the hypothesis H1, we applied the analysis on a larger subset of trace data with varying the degree of priority (e.g. weightage assigned to the quality-of-service attributes). Results from this analysis also show consistent trend in the observability scores. Hence, we infer that the metrics agree with the observability score calculated by our decision-making model which in turn support the hypothesis H1. We further investigate the behavior of an IoT system through more complex scenarios involving service composition.

Microservices can be combined to compose other services in a service composition process. Hence, it is ideal to identify the behavior of individual microservices such that it is possible to make a decision for the overall composed service and not only individual microservices. That is, we can then identify the microservices that contribute significantly to the deterioration in performance of the IoT system further supporting hypothesis (H1).

To further investigate and validate our observability score accuracy, we will evaluate the decision results at the microservice-level by composing a sample IoT system from the microservices available within the WS-Dream dataset. That is, we validated the accuracy of the observability scores through the analysis of individual microservice traces. Through this analysis, we would like to analyze the observability scores in more complex scenarios involving multiple microservices that work collectively to complete a business process. Through service composition, it is then possible to validate our observability scoring used for decision making using use cases that typically represent real-world scenarios.

For service composition, we assume that the services utilize a linear composition strategy where the output of one services becomes the input of the next service in the composition process. The observability scores are obtained from the average observability score of the 330 traces for each individual microservice employed in a service composition. This provides a more accurate overall observability score of a microservice whose performance may typically fluctuate over time. Hence, we cannot compare the scores to each other because the metrics and performance will be very different for each microservice. The scores are obtained by the normalization of values, then performing a decision analysis using the TOPSIS technique as described in the previous chapters. As we identify the accuracy of the decisions for individual traces within each microservice, we are using the average of observability score which has been calculated for each trace. Through this

strategy, we can depict and visualize the observability of a system which we shall demonstrate in the upcoming Chapter on building an IoT Prototype.

For the sample compositions use cases employed, we calculated another value delta Δ which denotes the percentage difference between the current actual observability score and the predicted observability score for each microservice. The formula used to calculate Δ is as follows:

$$\Delta \text{ (\%age difference)} = \frac{|\text{ Predicted Score} - \text{Actual Score }|}{\text{Actual Score}} \text{ x } 100 \quad (4.19)$$

4.3.4 Service Composition Use Case 1

We created a service composition comprising of ten microservices from the WS-Dream dataset for the analysis of the IoT system that will be generated by these microservices. The microservice sequence in this composition is shown in the Table 4.9.

Table 4.9. Service Composition Use Case 1 Results Analysis

Microservice ID	**Observability Score**	**Predicted Observability Score**	**Δ**	**Decision**
4026	0.127072	0.127072	0.000%	highRisk
4027	0.437594	0.437594	0.000%	mediumRisk
4029	0.510500	0.510500	0.000%	mediumRisk
402	0.129282	0.129354	0.557%	highRisk
4034	0.463554	0.463554	0.000%	mediumRisk
4036	0.447357	0.448526	0.261%	mediumRisk
4037	0.509163	0.509164	0.000%	mediumRisk
4038	0.712724	0.712715	0.001%	lowRisk
403	0.107069	0.107132	0.059%	highRisk
4040	0.574898	0.574914	0.003%	mediumRisk

In Table 4.9, the first column contains the ID of the microservice in the WS-Dream dataset. The second column represents the overall observability score of a microservice. The third column

represents the predicted observability score for the microservice which has been generated by the Prediction Model. The fourth column represents the delta values (Δ). Finally, the last column contains the decision about the state of the overall microservice based on current observability score (high-risk, medium-risk or low-risk).

The average observability score of the entire composite is 0.4019213 which indicates that it is categorized as medium risk. This is in accordance with the behavior of the microservices which compose the IoT system as the observability scores for the individual microservices fall near the medium-risk zone. As a result of this service composition use case, the individual microservices behavior does impact the overall behavior of the entire composed service. For example, 60% of the services in the service composition are moderate risk (e.g., medium) while there exists only one low risk microservice. Therefore, the observability score reflects the majority of the individual microservices observability score yielding moderate or medium risk.

4.3.5 Service Composition Use Case 2

We created a second service composition scenario comprising of ten microservices from the WS-Dream dataset for the analysis of the IoT system that will be generated by these microservices. The microservice sequence in this composition is very different than that of the service composition use case 1 in the sense that most of the microservices in the current service composition use case are clearly of low-risk zone as shown in the Table 4.10.

In Table 4.10, the first column contains the ID of the microservice in our dataset. The second column represents the overall observability score of a microservice. The third column represents the predicted observability score for the microservice which has been generated by the Prediction Model. The fourth column represents the delta values (Δ). Finally, the last column contains the

decision about the state of the overall microservice based on current observability score (high-risk, medium-risk or low-risk).

Table 4.10. Service Composition Use Case 2 Results Analysis

Microservice ID	Observability Score	Predicted Observability Score	Δ	Decision
4982	0.300128	0.299224	0.301%	highRisk
4873	0.896471	0.895011	0.163%	lowRisk
4874	0.843182	0.842602	0.069%	lowRisk
4875	0.941678	0.938820	0.304%	lowRisk
4876	0.838139	0.837457	0.081%	lowRisk
5097	0.202993	0.177619	12.499%	highRisk
4878	0.886533	0.886474	0.007%	lowRisk
4879	0.801750	0.800247	0.002%	lowRisk
487	0.567614	0.535189	0.057%	mediumRisk
4887	0.781362	0.781478	0.015%	mediumRisk

The average observability score of the entire composition is 0.705985 which is statistically significant and higher when compared to that of service composition use case 1. The average of 0.705985 represents a service composition that is categorized as low risk. Hence, the overall behavior of this composition is tightly coupled to the individual behavior of each microservice. In addition, the individual behavior of each microservice is reflective of its performance over a duration of time. Results from Table 4.9 show that the behavior of the microservices which compose an IoT system as the observability scores for many individual microservices fall in the low-risk zone. Therefore, this supports and proves our hypothesis that the ability to measure the observability of microservices can help us identify factors that may impact or influence the service level offering by an IoT system.

4.3.6 Service Composition Use Case 3

We created a service composition comprising of ten microservices from the WS-Dream dataset for the analysis of the IoT system that will be generated by these microservices. The microservice sequence in this composition is shown in the Table 4.11.

Table 4.11. Service Composition Use Case 3 Results Analysis

Microservice ID	**Observability Score**	**Predicted Observability Score**	**Δ**	**Decision**
114	0.073158	0.073190	0.044%	highRisk
1150	0.427464	0.427396	0.016%	mediumRisk
1151	0.672708	0.671266	0.214%	lowRisk
1154	0.096695	0.096764	0.071%	highRisk
1156	0.155218	0.155366	0.095%	highRisk
1157	0.456525	0.455496	0.225%	mediumRisk
1158	0.076956	0.076771	0.240%	highRisk
115	0.130054	0.130201	0.113%	highRisk
1162	0.230130	0.230402	0.118%	highRisk
1163	0.108129	0.107892	0.219%	highRisk

In Table 4.11, the first column contains the ID of the microservice in our dataset. The second column represents the overall observability score of a microservice. The third column represents the predicted observability score for the microservice which has been generated by the Prediction Model. The fourth column represents the delta values (Δ). Finally, the last column contains the decision about the state of the overall microservice based on current observability score (high-risk, medium-risk or low-risk).

The average observability score of the entire service composition scenario is 0.2427037 is statistically significant and lower when compared to that of service composition use cases 1 and 2, respectively. The average observability score suggests that this service composition

functionality of the system would be categorized as high-risk zone. Since a higher observability score represents better performance, this use case shows a low observability score for the service composition which is an indicator of the high risks associated with the microservices involved in this scenario. This is in accordance with the behavior of the microservices which compose the IoT system as the observability scores for many individual microservices fall in the high-risk zone.

Chapter 5
PROTOTYPE

The Distributed Observability Framework (DOF) can be extended and integrated as part of distributed tracing tools supporting the resilience and scaling of industry-level software applications. In this chapter, we describe a working prototype of front-end that we designed and developed for the purpose of demonstrating the usefulness of our proposed Distributed Observability Framework (DOF). We further use this prototype representing typical microservice composition scenarios required for the plurality of IoT systems at the application level and analyze the output from our DOF framework to test, support and validate our hypothesis (H1). In addition, we demonstrate how this framework can help in identifying bottlenecks in IoT systems and enable practitioners or software developers in selecting optimal or best alternatives when multiple microservices are available for performing backend tasks.

We have devised multiple use cases that vary in the magnitude of the number of composed services to illustrate the different applications of our framework and how it can be utilized or integrated into existing distributed systems in real-time. We first begin with introducing the sample proposed IoT system which we design and develop for further testing and analysis.

5.1 IoT System

We design a smart watch IoT system which involves a number of use cases that require service composition and distributed tracing capabilities. We will present the use cases of a user that utilizes a smart watch. Consider, for example, John representing an end user who has recently purchased a smart watch manufacture by company X. Assume that the system that is used by the smart watch utilizes service composition. That is, a number of different microservices are interacting with each

other for data collection, data retrieval among other features, as shown in Figure 5.1, to accomplish particular operations or tasks.

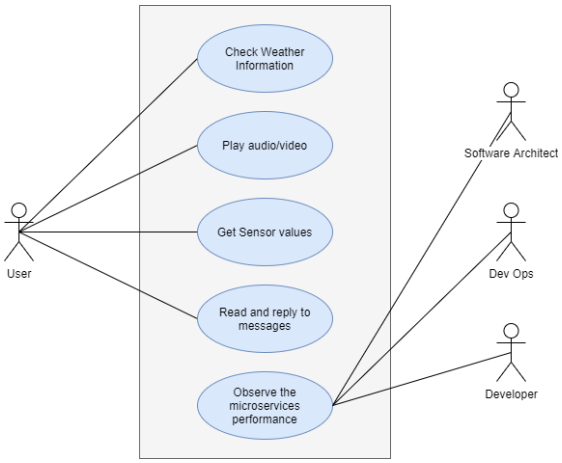

Figure 5.1. Use Case Diagram for Smart Watch IoT system.

John uses features offered through his smart watch for several different purposes throughout the day, including, but not limited to the following examples:

1. Reading data, calling other users and checking notifications from devices connected to the smart watch.
2. Performing actions or executing commands on the connected devices, such as setting reminders from an upcoming task.
3. Reading sensor values like heartbeat sensor, temperature sensor, location sensor, gyroscope, among others.
4. Booking movie tickets, flight tickets, among others.
5. Shop for groceries or order favorite coffee through his smart watch applications.

There can be many other activities that can be carried out by John throughout the day through his smart watch. All these activities can have some backend functionalities that required processing and are being fulfilled by a microservices architecture. Given that the smart watch application

typically provides a wide range of activities or functionalities, a significant amount of data will be generated by the smart watch and we assume this data will be stored in a stable repository. This data can be used for further processing. The data can then be used to analyze different microservices or different user workflows to find bottlenecks in the systems and predict the behavior of each functionality.

Based on our earlier explanation of the process of finding and predicting the behavior of such IoT systems in the previous chapters, we assume that our DOF framework is available through the smart watch application. Given that in a typical smart watch application, there may exist hundreds of microservices interacting with each other to deliver a wide range of functionalities, we will examine a subset of the possible use cases in the upcoming sections.

5.1.1 Set Call Reminder

One of the features that can be performed using a smart watch is setting a reminder to call a contact on John's phone. For reminding John to call the contact, the smart watch can have a composed microservice which in turn consumes other different microservices. The operations that a smart watch undertakes in this use case can be the following:

1. Retrieving John's reminders to remind him at the right time.
2. After reminding, if John wishes to call the selected contact, the watch can consumes another microservice to get his contacts from the linked account, and
3. Finally, it calls the contact.

We demonstrate how this use case can be utilized through our prototype in Figure 5.2.

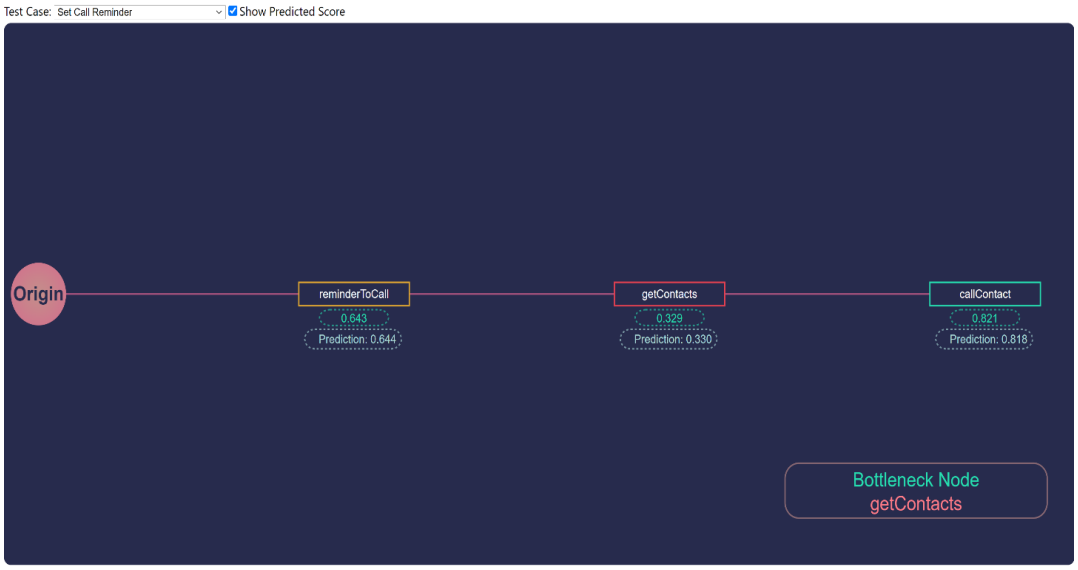

Figure 5.2. Setting Call Reminder from Smart Watch (UI).

As can be seen in Figure 5.2, the observability score of the microservices are displayed in the green bubble below the name of respective microservice for visualization. Below this bubble is the predicted observability score. These scores have been generated by the Distributed Observability Framework's Decision-Making Model and Prediction Model. The observability scores are generated at runtime based on the collected data. We utilize the WS-Dream for fetching this data to generate the observability scores in real-time. We selected microservices from the dataset and created the service with the results that were generated for those microservices for illustrating the service compositions scenarios that can be achieved through this IoT system.

Based on the observability score, a microservice is in high-risk state if the score is lower than 0.33 which is highlighted in red color on the dashboard or DOF's interface. If the observability score is between 0.33 and 0.66, then a microservice is considered to be in a medium-risk state which is highlighted in orange color. Similarly, if the observability score of a microservice is greater than 0.66, then the microservice is considered in low-risk state which is highlighted in green color.

As can be seen in Figure 5.2 through the front-end interface, DOF discovers the bottleneck node and display it in a box at the lower right corner. In this use case, the microservice with the lowest observability score is the ‘getContacts’. Hence, DOF designates this microservice as high-risk which indicates that this microservice is likely to have the largest impact in deteriorating the performance of the overall system. As a result, practitioners or software developers can then easily determine the causes or factors that are impacting the system’s performance through this service composition scenario.

Assume, for example, that the ‘getContacts’ microservice is experiencing a delay of 10ms due to which the sequence of events is paused until a response from this service is received. In this case, the ‘getContacts’ service is the bottleneck of the system since it is causing a degradation in the functionality’s performance due to an increase in overall response time. Hence, we wish to identify such bottlenecks for IoT systems such that it becomes possible to take preparatory actions in the event of a system failure in advance preceding the occurrence of the failure in an attempt to increase the resiliency and fault tolerance of IoT systems. We present a sequence diagram for this use case in Figure 5.3.

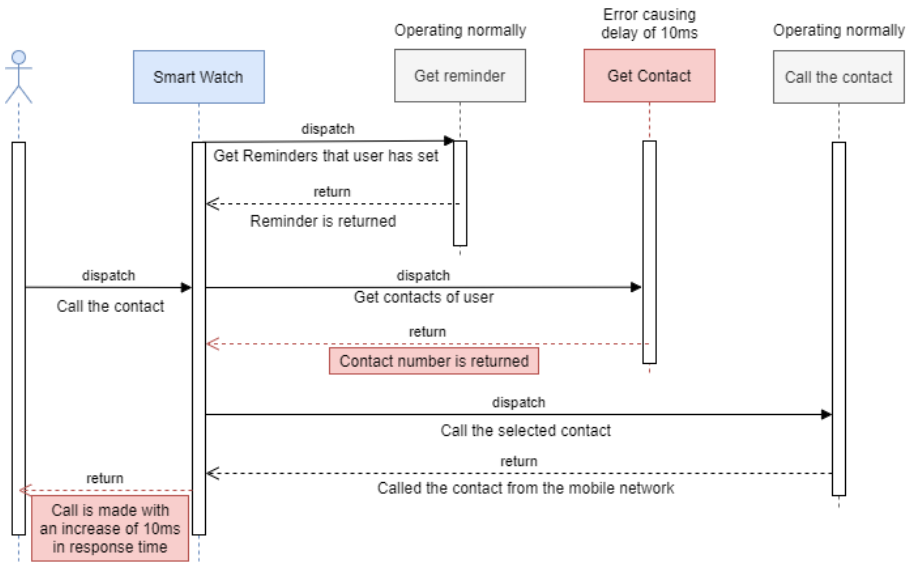

Figure 5.3. Setting Call Reminder from Smart Watch (Sequence Diagram).

5.1.2 Call My Phone

We illustrate the usefulness of our DOF framework through a second use case. For this use case, assume that John wishes the smart watch to call his phone. For delivering this functionality, the smart watch may need to undertake the following steps or operations:

1. Retrieving John's phone number from a linked cloud-based account.
2. Calling John's phone by using a service provider's microservice.

To implement this feature, we assume that the backend system provides different choices of selecting service providers which can be utilized for performing a phone call to John's phone number. The scenario with different microservices involved can be displayed on the UI as shown in Figure 5.4.

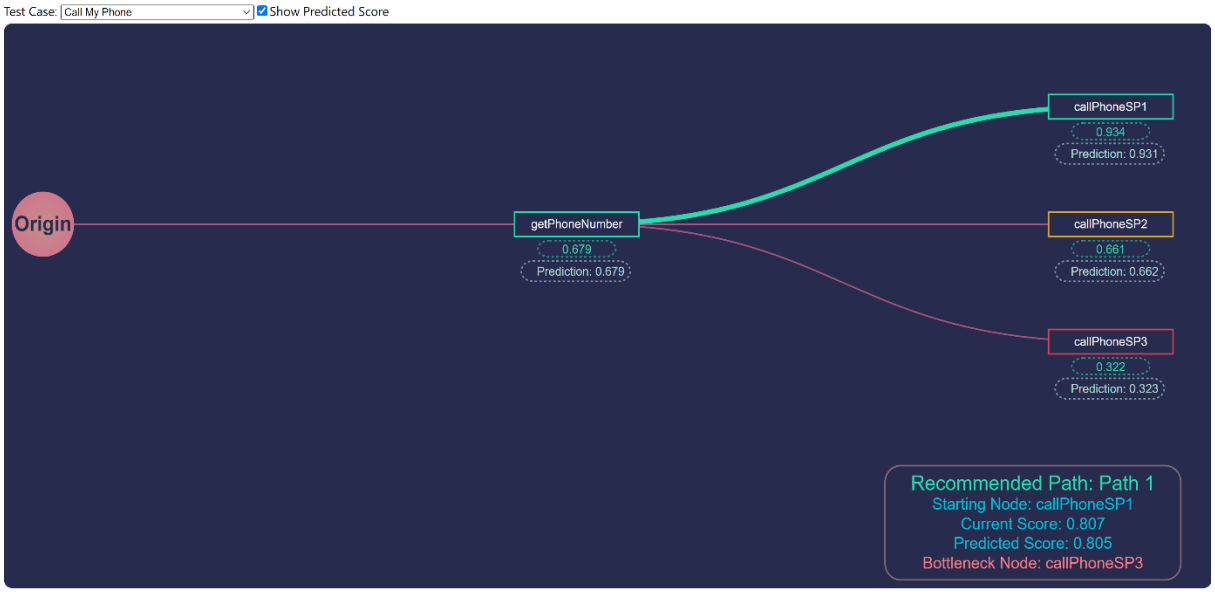

Figure 5.4. Calling user's phone from Smart Watch (UI).

As can be seen in Figure 5.4, assume that Service Provider 1's microservice is used for calling John's phone number and DOF determines its observability score being high with 0.934 whereas another service provider that can also perform the same functionality depicted as Service Provider

2's microservice has a medium-risk with an observability score of 0.661 and Service Provider 3's microservice is high-risk with observability score of 0.322. Due to this, our DOF system would be recommending the selection of Path 1 for the service composition since the average observability of the microservices on Path 1 has the highest amongst all available discovered alternatives. The current score in the box on the bottom-right represents the average of observability score of microservices on Path 1. The predicted score in the box is the average of predicted observability score of microservices on Path 1. The bottleneck microservice is pointed as the 'callPhoneSP3' since Service Provider 3's microservice has the lowest observability score in this case. The sequence diagram for the recommended path is shown in Figure 5.5.

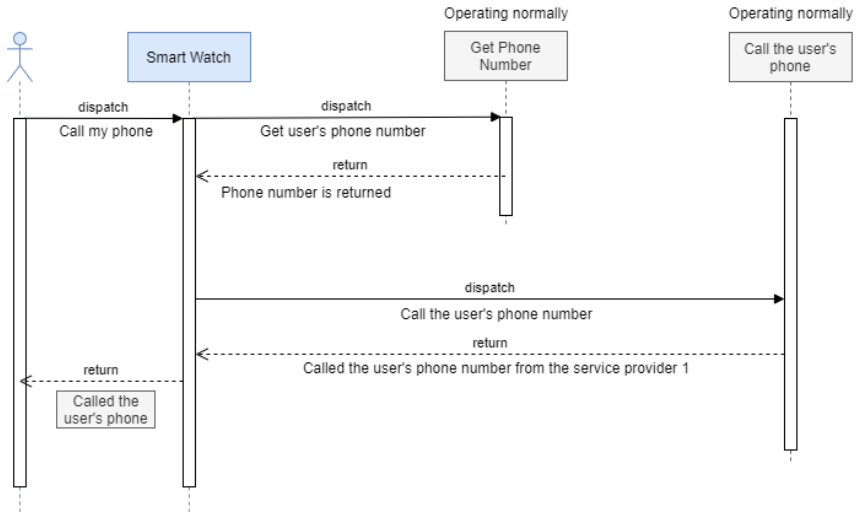

Figure 5.5. Calling user's phone from Smart Watch (Sequence Diagram).

This application can be integrated into existing cloud service providers (Google Cloud Platform, Microsoft Azure, Amazon Web Services, etc.) or distributed tracing tools (Zipkin, Jaeger, etc.) to deliver this functionality for IoT systems.

5.1.3 Check Weather Information

The next smart watch feature that we illustrate is to display weather information based on John's current location. Consider the use case where John checks the smart watch to get weather data. The watch then calls the appropriate external microservices chosen by the developers of company X in order to display the information. This process involves multiple microservices for delivering this functionality, including the following operations:

1. Retrieving John's IP address.
2. Retrieving the location (zip code) of John based on his IP address.
3. Retrieving weather information for John's locality based on zip code.

We assume that John has disabled his location services on smart watch and as a result the only possible way to access his geographic location is by first determining his IP address for the internet connection with a cellular service provider. To implement this feature, we consider that the backend system has access to a number of alternatives that are available through the use of microservices which can be used to create the service composition of the system in various ways to deliver this functionality. We illustrate these alternatives or available choices of this service level offering in Figures 5.6 and 5.7.

As presented in Figure 5.6, our DOF recommends the selection of microservices in Path 1 since this distributed path is associated with a higher average observability score than the alternative. However, from Figure 5.6, we can observe that the 'getIPAndZipCode' microservice in Path 2 has the highest observability score amongst all other microservices. Furthermore, Path 2 has the 'getWeatherSP2' which is a microservice which uses Service Provider 2. This microservice is determined as the bottleneck node since it has the worst observability score amongst all the microservices.

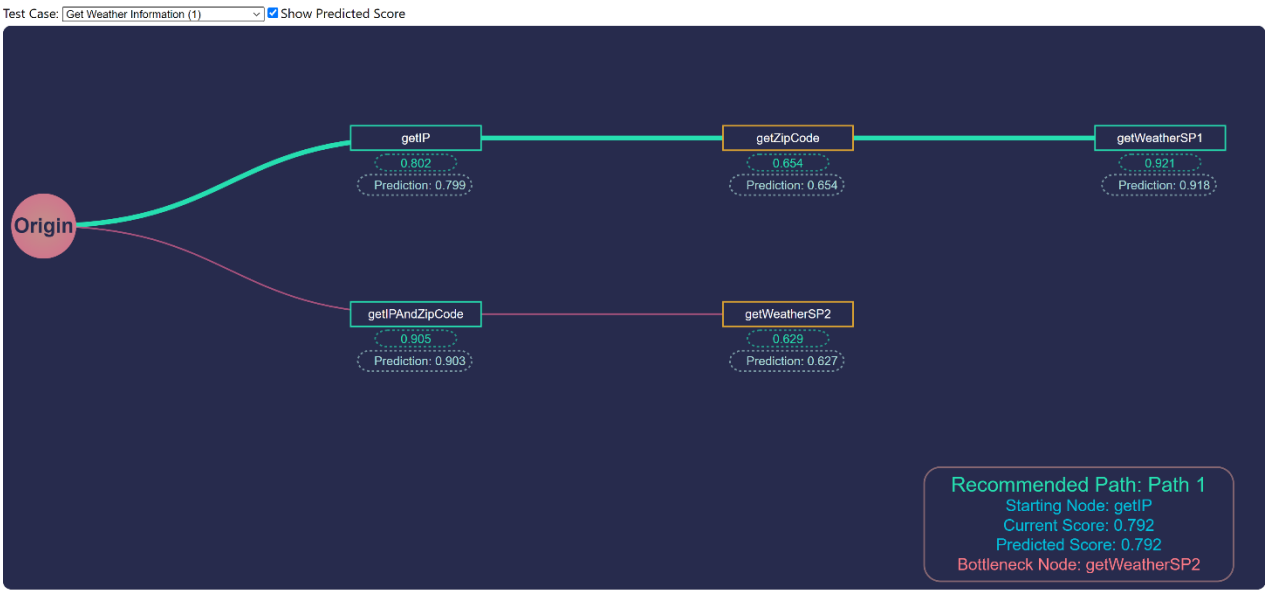

Figure 5.6. Service composition for weather information (UI) – Part 1.

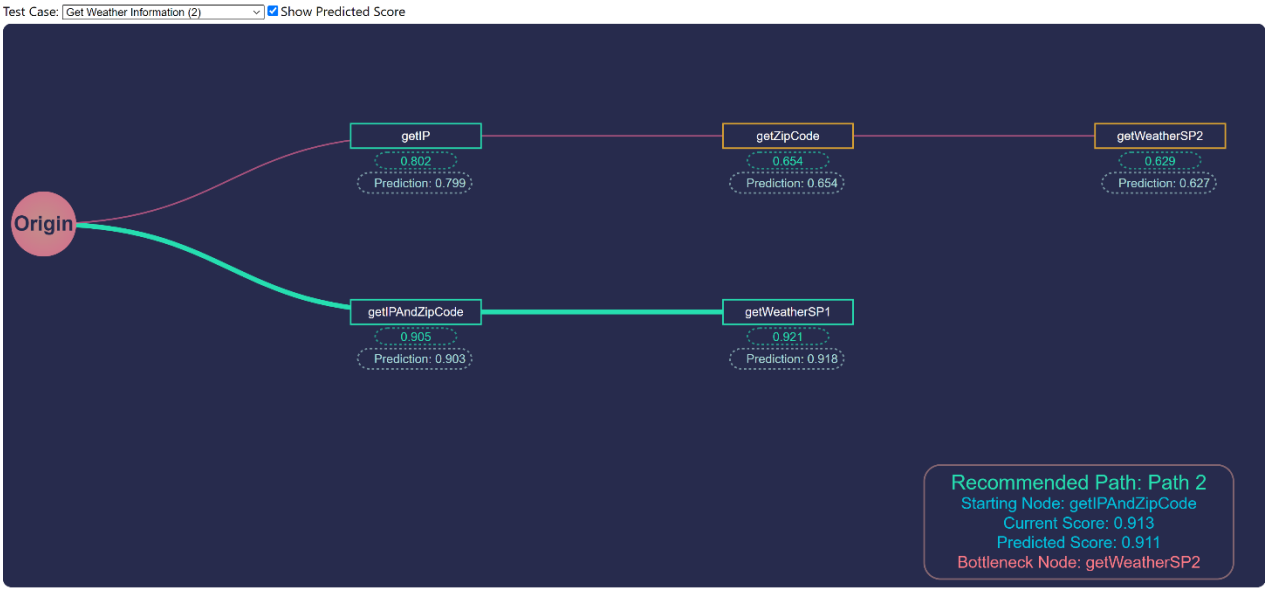

Figure 5.7. Service composition for weather information (UI) – Part 2.

Assume that decide to integrate the weather microservice by Service Provider 1 in combination with ‘getIpAndZipCode’ in Path 2. As presented in Figure 5.7, the DOF framework recommends the selection of the service Path 2 since the average observability score of Path 2 has increased by switching the weather microservice. This is because the ‘getIPAndZipCode’ and ‘getWeatherSP1’ are both low risk microservices. Hence, they shall lead to a superior

implementation of the overall IoT system. The observability score of Path 2 in Figure 5.7 is higher than that of Path 1 shown in Figure 5.6.

Through Distributed Observability Framework (DOF), practitioners or developers can utilize the interface to differentiate between service composition scenarios involving microservices and try different combinations to create the ideal service compositions for delivering a service functionality. Furthermore, through DOF, it is then possible to compare the predicted observability scores amongst the available microservice alternatives to determine whether a certain microservice is likely to deteriorate the overall performance of the system in the future prompting the need to take appropriate actions or decisions for their service composition. Hence, DOF helps enhance the observability of the overall behavior and identifies the sources of failure that are likely to cause a performance deterioration to the system. The sequence diagram for the collecting weather information use case with the best service composition alternative is demonstrated in Figure 5.8.

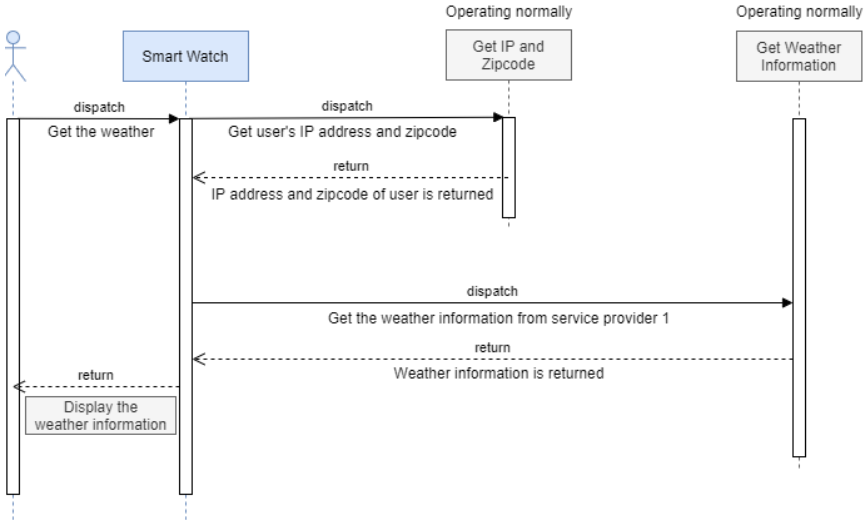

Figure 5.8. Getting Weather Information on Smart Watch (Sequence Diagram).

Chapter 6
CONCLUSION AND FUTURE WORK

In this research work, we investigated the usefulness of identifying the behavior of microservices through the use of observability methods that can be achieved using distributed tracing tools. To this extent, we introduced various methods that allow IoT systems to effectively measure and predict the observability of microservices. Through our observability model, we determine that it is possible to clearly identify sources of failure in a microservices' path and the root causes of the bottlenecks that impact their behavior and the overall behavior of IoT systems. In addition, as the number of microservices that compose major functionalities offered by existing IoT applications increases, the need for effective methods that can identifying the overall behavior of IoT systems over time becomes inevitable. Hence, it is essential to collect sufficient metrics that can enable frameworks to predict the observability of microservices and IoT systems. In this thesis, we introduced the Distributed Observability Framework (DOF) for the purpose of establishing observability measures for microservices that compose IoT systems. Through DOF, we are able to then make appropriate decisions for the IoT system and help prepare for system outages or select alternative service providers for service compositions scenarios or processes where applicable.

Our research into the observability of microservices addresses the three main research questions introduced in Chapter 1. Thus, we developed an Observability Model as part of our DOF framework for measuring the observability of microservices (RQ1, RQ2). In addition, we developed a multiple-criteria decision making technique based on TOPSIS to identify the bottlenecks that may occur within IoT systems which largely contribute to a deteriorating performance of the system over time (RQ1, RQ3). In addition, we developed a recommender module that recommends or selects the best service path for the composition for microservices

based on computed observability scores which are generated by the decision-making model of the DOF framework (RQ1, RQ2, RQ3). Furthermore, using DOF, it is possible to predict the observability of an IoT system (RQ2) based on historic observability data for individual microservices. In addition, we evaluated the framework using a real-world dataset (RQ2) and created the prototype with front-end for visualization of the service compositions, observability scores and the future applications of this research (RQ2, RQ3).

The Distributed Observability Framework (DOF) can have various applications of research across both academia and industry. The prototype designed for the DOF framework has been developed in the current phase and can be integrated into existing container management or orchestration tools such as Kubernetes on the Google Cloud Platform or Amazon's EC2 Container Service (ECS). In addition, the DOF framework can be extended as a plugin into existing distributed tracing and metric reporting tools such as Zipkin, Jaeger or Prometheus. Since we have tested our observability model with primary focus on IoT systems, further efforts need to be achieved to make the framework ready for the industry by testing on more variations of data for non-uniform and overlapping data which is generated from real-time IoT systems such as fitness trackers, smart speakers, smart home systems and smart watches, among many others.

BIBLIOGRAPHY

[1] Splunk, "Beginner's Guide to Observability," [Online]. Available: http://www.splunk.com/en_us/form/beginners-guide-to-observability.html. [Accessed 26 December 2020].

[2] S. Jaiswal, "(Tutorial) Markov Chains in Python," [Online]. Available: www.datacamp.com/community/tutorials/markov-chains-python-tutorial. [Accessed 26 December 2020].

[3] T. Hansen, "Monitoring Microservice-Based Cloud Applications Using Distributed Tracing," [Online]. Available: https://aws.amazon.com/blogs/apn/monitoring-microservice-based-cloud-applications-using-distributed-tracing/. [Accessed 30 December 2020].

[4] "Learn about Architecting Microservices-Based Applications on Oracle Cloud," Oracle, [Online]. Available: http://docs.oracle.com/en/solutions/learn-architect-microservice/index.html#GUID-1A9ECC2B-F7E6-430F-8EDA-911712467953. [Accessed 30 December 2020].

[5] A. M. Hammadi, T. S. Dillon and E. Chang, "Business service composability on the basis of trust," in *2009 3rd IEEE International Conference on Digital Ecosystems and Technologies*, Istanbul, France, IEEE, 2009, pp. 437-440.

[6] Google, "Site Reliability Engineering," [Online]. Available: http://landing.google.com/sre/sre-book/chapters/addressing-cascading-failures/. [Accessed 26 December 2020].

[7] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan and C. Shanbhag, "Dapper, a large-scale distributed systems tracing infrastructure," Google, 2010.

[8] New Relic, "About," New Relic, [Online]. Available: http://newrelic.com/about. [Accessed 26 December 2020].

[9] "Dynatrace," Wikipedia, [Online]. Available: http://www.en.wikipedia.org/wiki/Dynatrace. [Accessed 26 December 2020].

[10] Microsoft, "IoT Hub | Microsoft Azure," [Online]. Available: https://azure.microsoft.com/en-us/services/iot-hub/#overview. [Accessed 26 December 2020].

[11] OpenTelemetry, "About," [Online]. Available: http://opentelemetry.io/about/. [Accessed 26 December 2020].

[12] "OpenZipkin - A distributed tracing system," [Online]. Available: https://zipkin.io/. [Accessed 29 December 2020].

[13] "Jaeger: open-source, end-to-end distributed tracing," [Online]. Available: https://www.jaegertracing.io/. [Accessed 29 December 2020].

[14] "Splunk," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Splunk. [Accessed 26 December 2020].

[15] Lightstep features and product overview, "Lightstep features and product overview | Lighstep," [Online]. Available: https://lightstep.com/product. [Accessed 26 December 2020].

[16] "Distributed Tracing Working Group Charter," [Online]. Available: https://www.w3.org/2020/08/distributed-tracing-wg-charter.html. [Accessed 29 December 2020].

[17] Dynatrace, "Dynatrace joins the OpenTelemetry project," [Online]. Available: https://www.dynatrace.com/news/blog/dynatrace-joins-the-opentelemetry-project/. [Accessed 26 December 2020].

[18] New Relic, "OpenTracing, OpenCensus, OpenTelemetry, and New Relic," [Online]. Available: https://blog.newrelic.com/engineering/opentelemetry-opentracing-opencensus/. [Accessed 26 December 2020].

[19] "Jaeger and OpenTelemetry," [Online]. Available: https://medium.com/jaegertracing/jaeger-and-opentelemetry-1846f701d9f2. [Accessed 26 December 2020].

[20] "OpenTelemetry: Get Started With Lightstep," [Online]. Available: https://lightstep.com/developers/opentelemetry/. [Accessed 26 December 2020].

[21] "OpenTelemetry .NET SDK - distributed tracing and stats collection framework," [Online]. Available: https://github.com/open-telemetry/opentelemetry-dotnet#exporters-packages. [Accessed 26 December 2020].

[22] A. P. Bento, "Observing and Controlling Performance in Microservices," July 2019. [Online]. Available: http://jorge-cardoso.github.io/research/pdf_img/MSc_thesis_Andre_Bento.pdf. [Accessed 26 December 2020].

[23] B. Mayer and R. Weinreich, "A dashboard for microservice monitoring and management," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, Gothenburg, 2017.

[24] A. Noor, D. N. Jha, K. Mitra, P. P. Jayaraman, A. Souza, R. Ranjan and S. Dustdar, "A framework for monitoring microservice-oriented cloud applications in heterogeneous virtualization environments," in *IEEE 12th International Conference on Cloud Computing (CLOUD)*, 2019.

[25] Q. M. Ashraf, M. H. Habaebi and M. R. Islam, "TOPSIS-Based Service Arbitration for Autonomic Internet of Things," *IEEE Access*, vol. 4, pp. 1313-1320, 2016.

[26] A. Benveniste, E. Fabre and S. Haar, "Markov nets: probabilistic models for distributed and concurrent systems," in *40th IEEE Conference on Decision and Control (Cat. No.01CH37228)*, Orlando, FL, USA, 2001.

[27] Jin-LongWang, "Markov-chain based reliability analysis for distributed system," *Computers & Electrical Engineering*, vol. 30, no. 3, pp. 183-205, 2004.

[28] D. Khiatani and U. Ghose, "Weather forecasting using hidden Markov model," in *2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN)*, Gurgaon, India, 2017.

[29] A. Sahotra, "Intermittent Demand forecasting using Markov Chains," [Online]. Available: https://medium.com/analytics-vidhya/intermittent-demand-forecasting-using-markov-chains-d16fe3e2a0a3 . [Accessed 26 December 2020].

[30] M. Rezaeianzadeh, A. Stein and J. P. Cox, "Drought forecasting using Markov chain model and artificial neural networks," *Water resources management*, vol. 30, no. 7, pp. 2245-2259, 2016.

[31] C. Dabrowski and F. Hunt, "Using Markov chain and graph theory concepts to analyze behavior in complex distributed systems," in *The 23rd European Modeling and Simulation Symposium*, 2011.

[32] Z. Zheng, Y. Zhang and M. R. Lyu, "Investigating QoS of Real-World Web Services," *IEEE Transactions on Services Computing*, vol. 7, pp. 32-39, 2014.