

# **Alternative Visualisation of Distributed Tracing data in a complex, large-scale distributed system**

Noah Santschi-Cooney

Jason Quinlan

University College Cork

3rd April 2020

# Abstract

Modern Internet services are often implemented as complex, large-scale distributed systems. These applications are constructed from collections of software modules that could span many thousands of machines across multiple physical facilities. With the rise of modern Micro-service and Service-Oriented designs, traditional tooling used to monitor application behaviour is no longer viable, especially at scale.

To understanding the flow and life cycle of a unit of work performed in multiple pieces across various components in a distributed system, the concept of Distributed Tracing was born. Distributed Tracing was first introduced to the mainstream world in 2010 after the publication of Google's Dapper paper. Since then, various vendors have come out with their own Dapper-inspired services, most of them based off flame or timeline graphs.

The goal of this project is dual-faceted:

- Explore and research possible alternative uses and visualisation methods utilising data collected from distributed tracing clients.
- Implement one or more of the proposed alternatives.

# Declaration of Originality

In signing this declaration, you are confirming, in writing, that the submitted work is entirely your own original work, except where clearly attributed otherwise, and that it has not been submitted partly or wholly for any other educational award. I hereby declare that:

- this is all my own work, unless clearly indicated otherwise, with full and proper accreditation;
- with respect to my own work: none of it has been submitted at any educational institution contributing in any way to an educational award;
- with respect to another's work: all text, diagrams, code, or ideas, whether verbatim, paraphrased or otherwise modified or adapted, have been duly attributed to the source in a scholarly manner, whether from books, papers, lecture notes or any other student's work, whether published or unpublished, electronically or in print.

# Acknowledgements

# Contents

Abstract	i
Declaration of Originality	ii
Acknowledgements	iii
1 Introduction	1
1.1 Problem . . . . .	1
2 Background	3
3 Conclusion	4
Bibliography	5

# 1. Introduction

## 1.1 Problem

Within the last decade, the way modern applications are being built and deployed has changed dramatically. With the shift from collocation to cloud computing, virtual machines to containerization technologies, monoliths to micro-services and beyond, software developers have been able to adjust to the monotonical increase in internet traffic, shipping highly scalable, efficient and reliable software that meets the ever-demanding needs of their customers with the slew of emerging technologies.

While this shift has undoubtedly solved many issues with regards to scaling services in terms of both maintainability as feature sets increase and in keeping up with an every larger number of online users, it has introduced a whole new suite of problems that needed to be addressed in terms of reliability and application monitoring. With the splitting of monolithic applications into micro-services, the failure points are extended to issues in the network, including but not limited to network congestion, DNS resolution errors etc. Developers are ever more inclined to code failure resilience into their applications, falling back gracefully in apprehension of unforeseeable failures.

As these new distributed system architectures evolved and became ever more widespread, traditional application monitoring tools consistently fell short of providing developers and systems operators with the means to gain introspection into systems and their failures in production scenarios. Traditional monolithic systems often utilized logging and metrics to gain introspection into the application and for alerting on rules respectively. For such systems, these process-scoped measures often provided good insight into a system, correlating logs on their thread identifier/name.

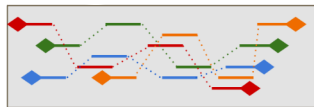
**“The Simple [Inefficient] Thing”**



**Basic Concurrency**



**Async Concurrency**



**Distributed Concurrency**

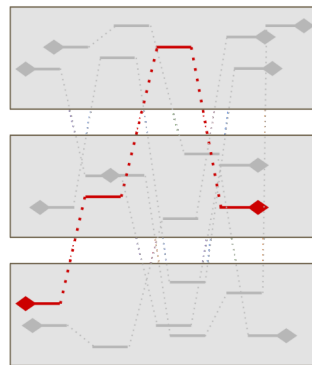


Figure 1.1: Evolution of concurrent systems

## 2. Background



### **3. Conclusion**

# Bibliography