Anomaly Detection & Root Cause Analysis In Distributed Systems

Project Proposal

Isala Piyarisi

w1742118 / 2018421

Supervisor: Guhanathan Poravi

Date: 3^{rd} November 2021

Department: Computer Science

Keywords: Cloud Computing, AIOps, Monitoring, Disaster Recovery

This project proposal is submitted in partial fulfilment of the requirements for the BSc(Hons) Computer Science degree at the University of Westminster.

Contents

List of Figures

List of Tables

List of Acronyms

IaaS Infrastructure as a Service

SRE Site Reliability Engineer

SLI Service Level Indicator

APM Application Performance Monitoring

MTTR Mean Time To Recovery

GAN Generative adversarial networks

HHMM Hierarchical hidden Markov model

FSL Few-shot Learning

SDLC Software Development Life Cycle

OOAD Object-oriented analysis and design

eBPF Extended Berkeley Packet Filter

1 Introduction

This document was made to provide the necessary context about one of the main pain points that arises when it comes to maintaining distributed systems and a course of actions that could be taken to reduce them. To do that author will first give a brief overview of the target domain and existing steps that have already been taken, then the author talks about shortcomings and improvements that can be made to them. Finally, the document will be concluded with how the author will approach the problem and try to solve it.

2 Problem Background

2.1 Cloud Computing

With an emergence Infrastructure as a Service (IaaS) like Amazon Web Services (AWS) and Google Cloud Platform (GCP) there is a big surge in organizations trying to outsource their computing needs to third parties (Rimol, 2021). This is mainly due to the elasticity given by all the cloud providers. Users can easily scale up and down their infrastructures within minutes without making any commitment and all the major providers, bill users on what you use are what you pay model because cloud provider manages all the underlying infrastructure users doesn't have to worry about problems like hardware failures. In contrast in a self-hosted setting if the user wanted one extra GB of memory than what's available it requires a lot of effort and cost to full fill that requirement.

2.2 Cloud-Native Applications

During the 90s and early 2000s, all the applications were made as a big monolith from a single code base (Spoonhower, 2018). Most of them were shipped as a single binary. Since those days applications were fairly simple this worked very well with little to no downsides. But when the 2010s came around there were a lot of specialized frameworks and programming languages and marketing teams wanted a lot of new futures quickly developed still maintaining reliability (Di Francesco et al., 2018; Mike Loukides, 2020). But if the code base of the application was stored in a single repository, developers have to go through a long process to review and test if the changes won't break the current system and developers are also limited by the framework and

programming language initial develops chosen for the project.

To tackle these problems there was a new way to develop applications was introduced, it's called "Microservices". The idea behind this concept is to break all the functionalities of big monolith applications into small individually scalable services and give ownership of each service to small teams of people who work separately. With this flow developers are free to use whatever tool they like to develop each service. Because these services are developed parallelly by different teams this increases the development velocity by order of magnitude (RedHat, n.d.).

As these services are relatively small and tailor-made to run on cloud environments it's very easy to take something that's running on the developer's local machine to the production cluster in a matter of minutes. This is mainly thanks to modern cloud-native tools like CI/CD pipelines which automatically build and test the code for them, which can save a lot of time spent just doing repetitive tasks which are prone to human errors (JetBrains, n.d.).

2.3 Monitoring Cloud-Native Applications

Even though cloud-native applications have a lot to offer when it comes to developer velocity and productivity, It has its fair share of issues. Most of these problems are linked to the sheer complexity of these systems and not having a proper way to monitor them (Zhitnitsky, 2019). All 3 major cloud providers provide a way to monitor these applications efficiently and some great open-source projects do this well, But to take full advantage of those systems, developers have to adapt their services to export all the vitals in a way the monitoring system understand. This works for the most part and this is what all the big companies are doing, even if it takes more developer time to in the end it's very crucial when it comes to disaster recovery.

But there is still a slight problem with this approach. Once the system starts to scale up to 100s of services number vitals that has to be monitored goes to 1000s and will require a lot of additional Site Reliability Engineers (SREs) and will have drop lot of non-crucial service vitals and derive abstract Service Level Indicators (SLIs) to make it **humanly** possible to understand what's going on.

3 Problem Definition

One of the main problems in monitoring microservices is the sheer number of data they generate. It's humanly impossible to monitor the metrics of all the services and it's hard for a single person to understand the entire system. To overcome this SREs use abstracted metrics called SLIs which measure the quality of the service at a higher level. SLIs will tell when there is an issue in the system, but it's very hard to understand where the actual problem is from it along. To understand the root cause of the problem SREs need to dig into Application Performance Monitorings (APMs) of all the services and go through the logs of each of the troubling services.

When the system consists of 100s or 1000s of services that are interdepended it's really hard to find where the actual issue is coming from and it may require the attention from all the service owners of failing services to go through the logs and APMs and identify the actual root cause of the failure. This could greatly increase the Mean Time To Recovery (MTTR) and waste a lot of developer time just looking at logs.

3.1 Problem Statement

Modern distributed systems are becoming big and complex so that when a failure happens it requires collaboration with a lot of people to find the root cause. Implementing a machine learning model which will watch over all the services and reacts to anomalies in real-time could greatly reduce the MTTR.

4 Research Motivation

Modern distributed systems generate tons of useful and not so useful telemetry data. As the system grows in demand and size, these telemetry data only get nosier and complex (Khononov, 2020). It's difficult for humans to make sense of all these data, especially if they don't have a lot of years of experience with the system. In the other hand, deep learning models thrive when it has a lot of data to learn from. As these models can be trained in computer-simulated environments they can learn concepts humans takes years to grasp within days (OpenAI, 2018; Silver et al., 2017). Finally, unlike humans a deep learning model can monitor a service 24x7 without taking any breaks which will not only prevent outages even before they happen, It could

be reduced MTTR because the issue can be detected way earlier than any human could do.

5 Existing Work

5.1 Anomaly detection

Citation	Technology summary	Improvements	Limitations
Du et al. (2018)	Tested most of common machine learning methods to detect anomalies and benchmarked them	• Used SLIs to monitored data • A lot of good metrics (input data) • Performance monitoring of services and containers	• Only be able to identify predetermined issues • Require a sidecar that includes a lot of overhead • Won't work with event-driven architectures (this is where most of the new systems are headed) • Uses Supervised learning and it's near impossible to find real-world data with labels
Kumarage et al. (2018)	The authors here are proposing a semi-supervised technique using a Variational Autoencoder to predict future time steps and calculate the difference between predicted and actual to detect anomalies.	• Due to the difficulty of finding labeled research data, they settled on using a semi-supervised technique. • Used randomized decision trees were utilized to select the most suitable features for each component.	• The model won't be easily transformable for other systems • If more new key features were added to the system it will require a total retraining

Kumarage et al. (2019)	Uses a bidirectional Generative adversarial networks (GAN) to predict future timesteps and uses MSE between prediction and real to determine the anomalies	Experimented using a GAN to detect anomalies rather than using conventional autoencoders	• Accuracy is around 60 • As this is a GAN-based system, it may take a lot of resources to run with production systems.

Table 1: Comparison of anomaly detection methods in distributed systems

5.2 Root cause identification

Citation	Technology summary	Improvements	Limitations
Gonzalez et al. (2017)	Detect failures in networks, using machine learning to generate knowledge graphs on historical data	• Build a predictable system • Automatic identification of dependencies between system events • Doesn't Need to rely on Domain experts • Generalized to different systems	• Limited to network issues • Even though the knowledge graph helped with visualization of the problem but still, people have to manually figure out what went wrong
Chigurupati and Lassar (2017)	Proposed a way to detect Hardware failures in servers using a probabilistic graphical model which concisely describes the relationship between many random variables and their conditional independence	• Find hidden meaning in values that seems random • Used a probabilistic approach to better understand the relationship between inputs and outputs • Gives all the possible root cause to a given problem	• Limited to hardware issues • Require support from domain experts • Can't account for unforeseen error

Samir and Pahl (2019)	This detects and locates the anomalous behavior of microservices based on the observed response time using a Hierarchical hidden Markov model (HHMM)	• Custom HHMM model • Self-healing mechanism • Focus on performance detection and identification at the container, node, and microservice level	• Input dataset scale is limited • Require a sidecar • Needs to predetermined thresholds
Wu et al. (2020)	Find Performance bottlenecks in distributed systems using an attribute graph to find anomaly propagation across services and machines	• Created a custom Faults Injection module • Uses an attribute graph to localize to faulty service • Application-agnostic by using a service mesh • Rely on service mesh to determine network topology • Uses unsupervised learning	• Only able to identify 3 types of issues • Looks only for performance anomalies • Use the slow response time of a microservice as the definition of an anomaly • Service meshes add a lot of overhead to systems • Required direct connection between services

Table 2: Comparison of root cause identification methods in distributed systems

5.3 Commercial products

Name	**Futures**	**Limitations**
Applied Intelligence by New Relic	• Metric forecasting. • Anomaly detection. • Alert grouping to reduce noise.	• Lack of explainability for certain classifications. • All the telemetry data need to be sent to a third party.

Watchdog by Datadog	• Monitor the metric data of the entire system from the background. • Monitor logging data. • Highlight relevant components affected by an issue.	• Announced in 2018 but is still at private beta. • Require code changes and tight integration with datadog platform. • Available demos about the system seems to be engineered for demonstration purposes.

Table 3: Comparison of commercial products for root cause analysis

6 Research Gap

After a literature survey author came conclusion finding a root cause of any failure within a distributed system is a very difficult issue due to it not having single output we can try to predict and most researchers have built their own simulation of a distributed system by themselves since there isn't any open dataset about monitoring data mainly because it could contain sensitive information.

Most currently established researches are done towards creating statistical models like clustering and linear regression. Even though these algorithms perform very well in small-scale systems, they struggle to keep up when the monitoring data become very noisy with scale. Another problem none of these papers properly addressed was constant changes to services. All most published research considers target services as static but in reality, these services can change even many times per day (Novak, 2016).

After talking with industry experts author concluded three main issues all had with using a machine learning model as monitoring agent Reliability, Interpretability, and Tunability. On reliability, experts said too many false positives will make operators lose faith in the system because it's gonna be another distraction to them. As the operators have to take critical decisions with the output of these models, it has been interpretable by humans (Ribeiro et al., 2016). Finally, this system should act more like a tool rather than a replacement to human operators, because no

matter machine learning models cannot compete with the context a human can handle.

7 Research Contribution

7.1 Domain Contribution

With this research, the author first tries to develop a **cloud-native solution to create a configurable microservices system**, So this research and future researches will have a standard environment to develop and evaluate their work. The author also hopes to build a lightweight and **low-overhead data collection pipeline** using Extended Berkeley Packet Filter (eBPF) to collect telemetry of target services without any instrumentation from the user.

7.2 Knowledge Contribution

One of the main problems with monitoring microservices systems is different services can be developed with different programming languages and frameworks and those can contain different levels of noisiness. So it's hard for a single model to detect anomalies in any service since some frameworks tend to use more resources while idle than others. So to address this author is trying to come up with an **encoding method** so the model can be trained to monitor one framework and those learning will still be valid for another framework. With those encoded data the author is hoping to develop a **convolutional autoencoder that will use unsupervised learning to spot out anomalies in a given data stream**. This may have better performance while using fewer resources convolutional layers are typically lightweight and good at pattern recognition (Oord et al., 2016). Finally, the author is planning to aggregate those predictions from the models into a pre-generated service graph and weigh it to **find all possible root causes**.

8 Research Challenge

Even though this project seems very straightforward and easy to implement from a high level, but it becomes tricky when attempting to reach targets defined in the section 12.1. For example, interpretability was one most requested feature from industry experts and a must-have trait for mission-critical systems (Ribeiro et al., 2016). But it was left out of the project scope due to its complexity especially when it comes to an **undergraduate project**. Other than that following are a few of the more difficult challenges the author is expected to face while conducting the research.

- **Highly seasonal and noisy patterns** - Monitoring metrics on microservices on production tends to have very unpredictable patterns depending on the traffic that's sent to the service. The amount of traffic sent will depend on several external factors that are hard to determine. Modeling both temporal dependencies and interdependencies between monitoring data into a single graph will be very difficult and require a lot of fine-tuning and data engineering.
- **Overhead** - Modern deep learning models can solve any problem if we could give it an unlimited amount of data and processing power but In this case, models need to optimize for efficiency over accuracy since having a monitoring system that consumes a lot more resource than the actual target system isn't effective.
- **Fit into Kubernetes eco-system** - Kubernetes has become the de-facto standard to managing distributed systems (IBM, 2021). So the author is planning to create a Kubernetes extension that will bridge the connection between monitored service and monitoring model as shown in the figure 1. But Kubernetes itself has a very steep learning curve, even the original developers themselves admitted it's too hard complex for beginners Anderson (2021).

9 Research Question

RQ1: How can a machine learning model improve MTTR in a distributed system?

RQ2: What is the most efficient way to present raw data monitoring to machine learning model?

RQ3: What will be the most ideal machine learning model to uncover anomalies in a microservice?

RQ4: What are the methods that can be used to evaluate a root cause prediction system?

10 Research Aim

The aim of this research is to design, develop and evaluate a toolkit to help system operators to reduce the MTTR when the system is experiencing an anomaly by using a machine learning model investigating all the services in the system and highlighting the most probable root causes in order, So the operators don't have to find a needle in a haystack.

To achieve this author tries to create a single model that can monitor all the vitals of a given service and output an anomaly score in any given time window. The author is hoping

to make it generalized enough so operators can take the same model and deploy it with other services and the model will adopt the new services with Few-shot Learning (FSL) (Wang et al., 2020). To do this author is trying to create a data encoding technique to represent monitoring data in a programming language or framework independent way.

Finally, the author is hoping to develop a playground that easily simulates a distributed system within a Kubernetes cluster so the create system can be tested and evaluated properly and future researches on this domain will have to benchmark framework to evaluate their work.

11 Research Objectives

Research Objectives	**Explanation**	**Learning Outcome**
Problem identification	When selecting the problem author wanted to pursue, they had 3 main goals. 1. The problem domain should be something they enjoy working in. 2. At the end of the research should have done a meaningful impact on the target domain, both in the theoretical and practical aspect, 3. It should be challenging to achieve and results should speak about themselves.	LO1
Literature review	Conduct a Literature review on root cause analysis to, • To find the current methods used to anomaly detection and localization. • Uncover issues with current approaches. • Understand how advancement in other related domains can apply to this domain.	LO3, LO4, LO6

Developing an evaluation framework	During the literature survey, one problem the author identified was there isn't a uniform dataset when it comes to training and evaluating models to detect anomalies in microservices. Most of the researchers used private datasets to train and test their work. To address this author is developing, • A tool that can easily simulate a distributed system in a cloud-native setting. • A tool inject anomalies into the running services.	LO7
Publish a paper about that playground	The author is hoping to publish a paper about the above-mentioned tool so the future researchers will have a unified way to train, test, and benchmark their system without having to reinvent the wheel again and again.	LO7
Data gathering and analysis	In order to create model to detect anomalies the author will, • Simulate distributed system. • Simulate traffic inside the system • Collect monitoring data while it's running	LO7
Developing encoding method	As mentioned in the section 7.2 these services will report very different values even at idle. To normalize data from all the services to one format author will, • Evaluate current data encoding methods like Zhang et al. (2019). • Find the best one fit and optimize it to this use case. • Test if there is any improvement by using this method.	LO2, LO5, LO7

Developing the model	According to Kumarage et al. (2019) Autoencoders tend to perform best when it comes to anomaly detection. But during the literature survey it was raveled Conversational Autoencoders weren't tested. So author tries to develop a Conversational Autoencoders and test how it will perform.	LO2, LO5, LO7
Testing and evaluation	Following things will be tested during the testing phase, • How will the system classify long-term fluctuations. • How will the system classify short-term fluctuations. • Can the system understand the mapping between core metrics like CPU and Memory usages. • Accuracy of fault detection. • Accuracy of root cause localization.	LO8, LO9
Integration	Having a fancy model doesn't add means anything if it's very hard to use in a real system. So the author is hoping to develop a Kubernetes extension that will map the model with any service given by the user.	LO7

Table 4: Research objectives

12 Project Scope

From the literature survey and talking with industry, experts author found many issues they can address when developing the system, but some of those problems like interpretability on autoencoder (Ribeiro et al., 2016) are hard to solve by someone at a level of an undergraduate. As this project is done by one developer in less than one year, it won't be possible to create a fully functional monitoring platform like Datadog or New Relic. The Force of this project is to see if the author can develop a single model that can monitor all kinds of services after transfer learning with few examples.

12.1 In-scope

Following are the main forces of this project

- Evaluation Framework
  - Ability to create service mesh out using Kubernetes native resources.
  - Each service has the ability to simulator predefined error types.
  - Service mesh can be made up of services written in different programming languages and frameworks.
  - Built-in method to run stress tests.
- Monitoring System
  - Low overhead data collection pipeline to collect service telemetry.
  - Reliability system which generate fewer false positives so it won't overwhelm the operators and false negatives will be caught by the main monitoring system.
  - Optimized models to have fairly small memory footprint and a CPU overhead.
  - Well generalized model which will be able to deploy with completely new services and it will learn to adapt the new system.

12.2 Out-scope

Follow will not be covered during this project

- Evaluation Framework
  - Support for every major language and framework.
  - Working outside of Kubernetes eco-system.
- Monitoring System
  - Interpretability - Describing a behavior of autoencoder is a difficult task that won't be covered during the project.
  - System won't be trained against data from a real production system due to the lack of public datasets.
  - System won't have very high accuracy, as this will be the first line of defense this will try to avoid false positives to prevent adding more noise to alerting systems.
  - Automatically identify system topology.
  - This will not be a drop-in replacement for existing monitoring systems, rather this will work with existing monitoring systems to reduce the MTTR.

12.3 Prototype Feature Diagram

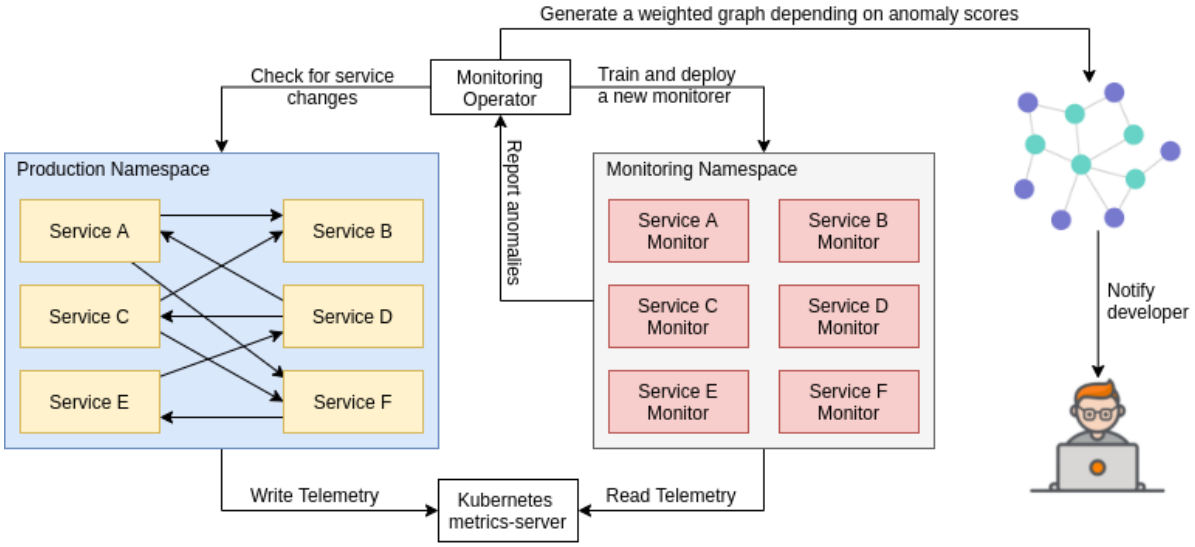

Figure 1: Prototype feature diagram (self composed)

13 Research Methodology

Research Philosophy	Mainly, there are four research philosophies, Pragmatism, positivism, realism, and interpretivism. It explains the belief and the research is done. After doing an in-depth study about research philosophies, the author decided on following **Pragmatism** as the research philosophy because the author believes there is no one way to solve the problem this research is tried to address and the goal of this research is to solve a practical problem faced by SREs. (Munim (2019), Dudovskiy (n.d.))
Research Approach	Although the inspiration for the research came from an observation of the real world. The author is using **deductive reasoning** to approach the problem. After the problem was identified the author looked for existing work found few theories on the domain. Then the author found few flaws in these methods thought of a way to address them with different approaches. At the end of the research other hopes to implement these new approaches and observe their outcome.

Research Strategy	The research strategy will be used to answer the research questions. In this project, the author will use **experimenting, interviews, and surveys** to provide answers to research questions.
Research Choice	During this research project, the author is planning to build a very generalized solution to predict anomalies. So to achieve this, a **quantitative** dataset will be used to train the model while a **qualitative** data set will be used for evaluate it. So the data for this research will be collected by using the **Mixed method**.
Time zone	This project needs to be completed within 8 months, so a **cross-sectional** time horizon will be used to collect data to complete the project.

Table 5: Research methodology selection

14 Development Methodology

Even though this project has few clearly defined requirements, designing and developing them will require an iterative model as there isn't a single best way to develop this and the author will be experimenting with different techniques. Thus the author decides on using **prototyping** as the Software Development Life Cycle (SDLC) Model for this project.

14.1 Design Methodology

To design the system diagrams for this project Object-oriented analysis and design (OOAD) methods will be used. OOAD make it easier to design the system iterative and this complement the choice SDLC method Prototyping.

14.2 Evaluation Methodology

During the literature, the survey author concluded that there are not any specific evaluation metrics for the root cause analysis system other than accuracy and f1 score, and there are not any publicly available datasets or systems to benchmark against. Base-level benchmarks will be carried out to compare the proposed system with the existing ones.

14.3 Requirements Elicitation

As the results of this project will be mostly used by SREs and system administrator the author is hoping to talk with few of the experts in the respective fields to get a better idea on what are the things to be expected from a system like this. Moreover as mentioned in 12.2 this system is not designed to entirely replace existing monitoring systems, So the author is hoping to research about production monitoring systems and their workflows to understand how the proposed system could seamlessly integrate them.

15 Project Management Methodology

To manage task of this project authors decide to use **Agile PRINCE2**. Agile PRINCE2 built upon waterfall method which works best for projects with fixed deadlines and requirements with the added benefit of having regulated inputs and outputs (Chappell, 2021).

15.1 Deliverables

Deliverable	Date
Draft Project Proposal A draft version of this proposal	02nd September 2021
A working beta of MicroSim MicroSim is a tool that simulates a distributed system within a Kubernetes cluster. This tool will be used to test and evaluate the final version of this project	15th September 2021
Research Paper about MircoSim MicroSim could have various other use-cases and could help in the development of this research domain. So the author is planning to release it as an open-source project with paper so future research and benefits from this.	16th October 2021
Literature Review Document The Document explaining all the existing tools and published researches on the domain	21st October 2021
Project Proposal The final version of this project proposal.	04th November 2021

Software Requirement Specification The Document all the key requirements that are gonna get address with this research	25th November 2021
Proof of Concept Unoptimized prototype with all the main features working	06th December 2021
Interim Progress Report (IPR) The document explaining all the preliminary findings and the current state of the project	27th January 2022
Test and Evaluation Report A document with results of the project and conclusion made from those tests	17th March 2022
Draft Project Reports The draft version of the final thesis	31st March 2022
Final Research Paper A paper with results about this project	14th April 2022
Final Project Report Finalize version of the thesis	28th April 2022

Table 6: Deliverables and due dates

15.2 Schedule

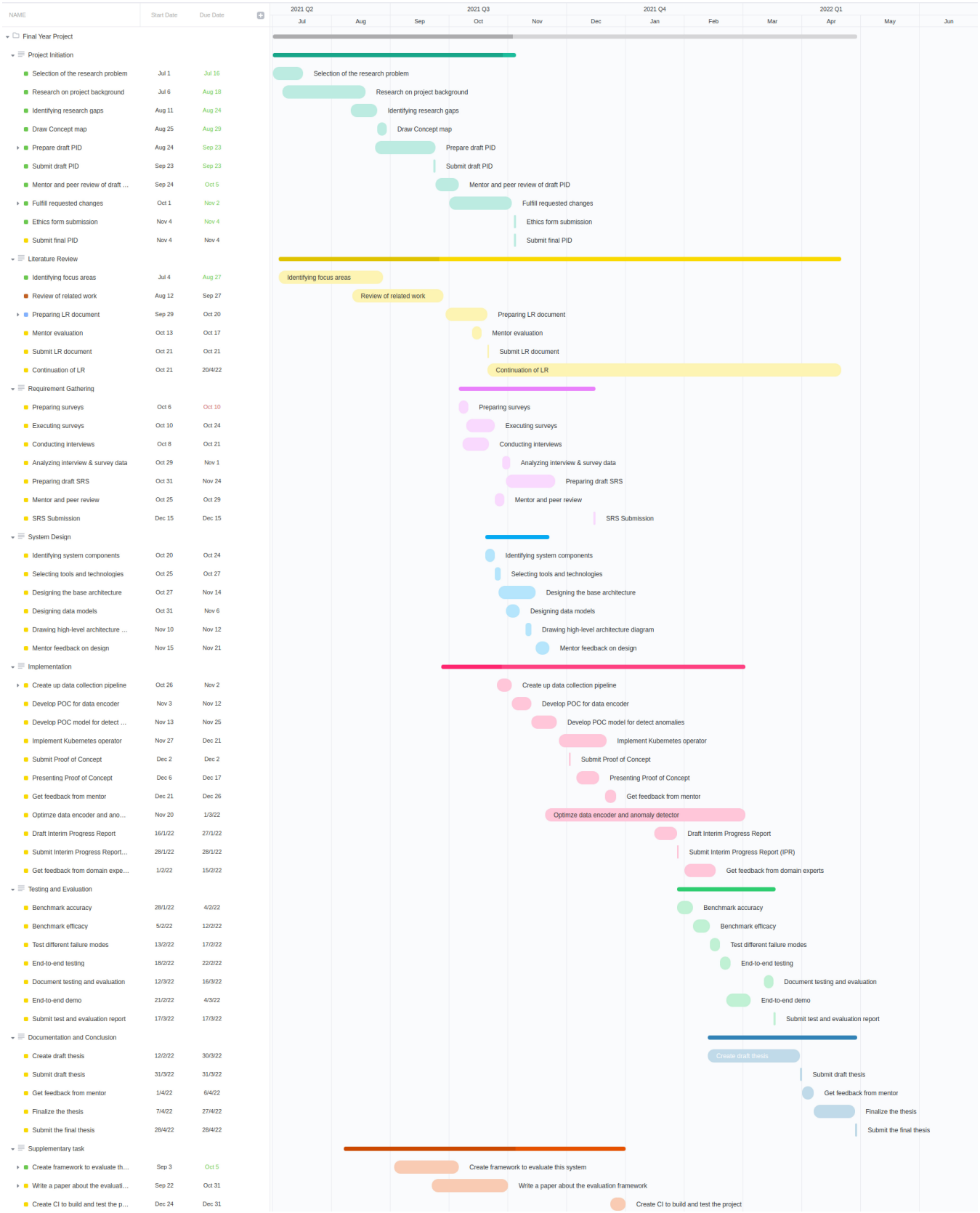

Figure 2: Defined gantt chart for the project (self composed)

15.3 Resource Requirement

15.3.1 Software Requirements

- **Ubuntu / Arch Linux** - Since this project will use eBPF as a dependency it will require a Linux kernel based operating system.
- **Python / R** - This project has a data science. So using a language with good data science eco-system will make this process easier.
- **GoLang / Rust** - While GoLang has official client library made by Kubernetes developers themselves, kube-community has developed an excellent alternative in Rust.
- **K3d / Minikube** - To create a Kubernetes cluster locally for development and testing.
- **Jetbrain IDEs / VS Code** - IDE provides lot of tools that will help developing complex project like this easily.
- **Google Docs / Overleaf** - To create documation about the project the author can use a usual editor like Google Docs or declaratively tool like Overleaf which use coding like style to format the document.
- **Google Drive / Github** - Offsite location to backup the codebase and related documents.
- **ClickUp / Notion** - To manage the project and keep track of things to be done.

15.3.2 Hardware Requirements

- **Quad-core CPU with AVX support** - AVX is a CPU instruction set which is optimze for vector operations. Having an AVX supported CPU could reduce the model inference time.
- **GPU with CUDA support and 2GB or more VRAM** - Both Tensorflow and Pytorch depend on CUDA for hardware-accelerated training. Training on GPU could save a lot of time increases the number of trial and error iterations that could be done.
- **16 GB or more Memory** - Running a microservices simulation locally will consume a lot of memory and while testing models will get loaded into RAM.
- **At least 40GB disk space** - To store the dataset, models docker containers while developing the project.

15.3.3 Skill Requirements

- **Experience working with Kubernetes** - The author will be developing a Kubernetes extension so they need to know the inner workings of Kubernetes.
- **Data engineering** - Developing a data encoding technique requires a lot of knowledge in how to manipulate a given dataset.

- **Model engineering** - Creating model from ground up is difficult task. So the author needs to have an in-depth idea about a machine learning framework and how different layers in the model work in order to fit them properly.

15.3.4 Data Requirements

- **Monitoring dataset** - This dataset can be collected using MicroSim tool author plan to develop to simulate distributed system.

15.4 Risk Management

Risk Item	Severity	Frequency	Mitigation Plan
The hypothesis the research is based on is wrong	5	1	Present the findings and explain why the hypothesis was wrong
Failure in work computer	4	3	Daily backup work the work to a cloud platform
Lack of domain knowledge	2	3	Talk to a domain expert, Do more research
Models not generalizing	3	4	Explore different methods, Try cleaning up the dataset more
Dataset quality is not up to the standard	4	1	Use a method used in related researches to create a new dataset
Running out of time	1	2	Following a thorough work schedule
Getting sick and unable to work for few days	3	3	Keeping few days of a buffer period before deadlines

Table 7: Risks and mitigations

References

Anderson, T. (2021), 'Google admits kubernetes container tech is so complex, it's had to roll out an autopilot feature to do it all for you • the register', https://www.theregister.com/2021/02/25/google_kubernetes_autopilot/. (Accessed on 09/14/2021).

Chappell, E. (2021), 'What are the top 8 project management methodologies?', https://clickup.com/blog/project-management-methodologies/#36-7-what-is-the-prince2-methodology. (Accessed on 09/22/2021).

Chigurupati, A. and Lassar, N. (2017), Root cause analysis using artificial intelligence, *in* '2017 Annual reliability and maintainability symposium (RAMS)', IEEE, pp. 1–5.

Di Francesco, P., Lago, P. and Malavolta, I. (2018), Migrating towards microservice architectures: an industrial survey, *in* '2018 IEEE International Conference on Software Architecture (ICSA)', IEEE, pp. 29–2909.

Du, Q., Xie, T. and He, Y. (2018), Anomaly detection and diagnosis for container-based microservices with performance monitoring, *in* 'International Conference on Algorithms and Architectures for Parallel Processing', Springer, pp. 560–572.

Dudovskiy, J. (n.d.), 'Pragmatism research philosophy', https://research-methodology.net/research-philosophy/pragmatism-research-philosophy/. (Accessed on 09/08/2021).

Gonzalez, J. M. N., Jimenez, J. A., Lopez, J. C. D. et al. (2017), 'Root cause analysis of network failures using machine learning and summarization techniques', *IEEE Communications Magazine* **55**(9), 126–131.

IBM (2021), 'What is container orchestration?', https://www.ibm.com/cloud/learn/container-orchestration. (Accessed on 09/14/2021).

JetBrains (n.d.), 'What are the benefits of ci/cd?', https://www.jetbrains.com/teamcity/ci-cd-guide/benefits-of-ci-cd/. (Accessed on 08/26/2021).

Khononov, V. (2020), 'Untangling microservices or balancing complexity in distributed systems', https://blog.doit-intl.com/untangling-microservices-or-balancing-complexity-in-distributed-systems-7759987d4 (Accessed on 08/31/2021).

Kumarage, T., De Silva, N., Ranawaka, M., Kuruppu, C. and Ranathunga, S. (2018), Anomaly detection in industrial software systems-using variational autoencoders., *in* 'ICPRAM', pp. 440–447.

Kumarage, T., Ranathunga, S., Kuruppu, C., De Silva, N. and Ranawaka, M. (2019), Generative adversarial networks (gan) based anomaly detection in industrial software systems, *in* '2019 Moratuwa Engineering Research Conference (MERCon)', IEEE, pp. 43–48.

Mike Loukides, S. S. (2020), 'Microservices adoption in 2020 – o'reilly', https://www.oreilly.com/radar/microservices-adoption-in-2020/. (Accessed on 09/22/2021).

Munim, Z. H. (2019), 'Philosophy of science | four major paradigms', https://www.youtube.com/watch?v=n8B50HJrAv0. (Accessed on 09/08/2021).

Novak, A. (2016), 'Going to market faster: Most companies are deploying code weekly, daily, or hourly', https://newrelic.com/blog/best-practices/data-culture-survey-results-faster-deployment. (Accessed on 09/04/2021).

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. and Kavukcuoglu, K. (2016), 'Wavenet: A generative model for raw audio', *arXiv preprint arXiv:1609.03499* .

OpenAI (2018), 'Openai five', https://blog.openai.com/openai-five/.

RedHat (n.d.), 'Understanding cloud-native apps', https://www.redhat.com/cloud-native-apps. (Accessed on 08/26/2021).

Ribeiro, M. T., Singh, S. and Guestrin, C. (2016), "why should i trust you?" explaining the predictions of any classifier, *in* 'Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining', pp. 1135–1144.

Rimol, M. (2021), 'Gartner says worldwide iaas public cloud services market grew 40.7% in 2020', https://www.gartner.com/en/newsroom/press-releases/2021-06-28-gartner-says-worldwide-iaas-public-cloud-services-market-grew-40-7-per

Samir, A. and Pahl, C. (2019), Dla: Detecting and localizing anomalies in containerized microservice architectures using markov models, *in* '2019 7th International Conference on Future Internet of Things and Cloud (FiCloud)', IEEE, pp. 205–213.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. et al. (2017), 'Mastering the game of go without human knowledge', *nature* **550**(7676), 354–359.

Spoonhower, D. (2018), 'Lessons from the birth of microservices at google', `https://dzone.com/articles/lessons-from-the-birth-of-microservices-at-google`. (Accessed on 09/22/2021).

Wang, Y., Yao, Q., Kwok, J. T. and Ni, L. M. (2020), 'Generalizing from a few examples: A survey on few-shot learning', *ACM Computing Surveys (CSUR)* **53**(3), 1–34.

Wu, L., Tordsson, J., Elmroth, E. and Kao, O. (2020), Microrca: Root cause localization of performance issues in microservices, *in* 'NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium', IEEE, pp. 1–9.

Zhang, C., Song, D., Chen, Y., Feng, X., Lumezanu, C., Cheng, W., Ni, J., Zong, B., Chen, H. and Chawla, N. V. (2019), A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data, *in* 'Proceedings of the AAAI Conference on Artificial Intelligence', Vol. 33, pp. 1409–1416.

Zhitnitsky, A. (2019), '5 ways you're probably messing up your microservices | overops', `https://www.overops.com/blog/5-ways-to-not-f-up-your-microservices-in-production/`. (Accessed on 08/26/2021).