Comparison of Service Mesh Technologies

Jose Luis Povedano Poyato - 2403203P

March 31, 2023

1 Status report

1.1 Proposal

1.1.1 Motivation

Microservices are an approach to structure applications as a collection of loosely connected services. Each service focuses on a specific task, simplifying building and scaling applications as different functionalities become independent from each other. This makes microservices a very popular approach for software development, with many large companies like Amazon, Netflix and eBay relying on them.

Service Mesh Technologies (SMTs) are technologies used to control service-to-service communication over a network. SMTs are used to simplify how the microservices that form an app communicate. They function independently from the microservices, abstracting network features from the service development. However, using SMTs could mean a higher latency and resource consumption, as they are using part of the allocated resources. This project aims to compare the performance of different SMTs in different scenarios. This should help developers decide if they want to use SMTs and which one could work best for their needs.

1.1.2 Aims

This project aims to compare different service mesh technologies. The objectives are to identify suitable attributes to compare SMTs, then devise benchmarks for quantitative comparisons. The goal is to develop a benchmark that measures the performance and resource usage of various Kubernetes Service Mesh Technologies in many different scenarios.

1.2 Progress

- Investigated previous approaches at comparing Kube SMTs and designed new experiment architecture
- Language and frameworks for micro-services selected: Python and Flask to create services
- Selected technologies to measure latency and resource usage: Apache Ab, Prometheus and Grafana
- Set up Docker and Kubernetes (KinD distribution) in Windows, WSL and Supervisor's Server
- Tested how bare kubernetes cluster communication between services works
- Coded and deployed a counter microservice that redirects requests to other deployments of itself so we can simulate differently sized microservice networks in kubernetes

- Automatised request load generation, so they are automatically sent to counter and latency is recorded

1.3 Problems and risks

1.3.1 Problems

- Inexperience. Prior to this project I had no experience with Docker or Kubernetes. I had some experience with other container systems through an industry placement, however, those containers were internal technology and function quite differently from these. Furthermore, I had no experience with SMTs, so I had to do a lot of research about different distributions and procedures.
- Technology incompatibilities. My Windows OS was not compatible with the required features for the Istio SMT. My local distribution of Ubuntu using WSL, was not compatible with docker. My supervisor provided me with a server, where everything is working so far. The compatibility issues led to restarting set ups multiple times, with different issues each time which delayed progress.
- Installing SMTs is more complicated than I expected. Some SMTs (e.g. Istio) have different distributions, with different features and installation procedures, which can affect performance and usage. There are many more options to consider than originally expected.

1.3.2 Risks

- There is a wide range of SMTs to consider for the benchmark. So far have decided on Istio, Linkerd and Consul, but there are many others. **Mitigation:** By start of next semester narrow or expand SMT selection depending on available time and progress.
- Installing SMTs is more complicated than I expected. Based on my experience with the Istio SMT, there is not a straightforward process. They can have multiple different distributions, with a different installation process and overall functionality. The SMT performance can depend on the distribution chosen. This leads to a more time consuming research and installation. **Mitigation:** Allocate more time to the SMT installation process, consider reducing the amount of SMTs used.

1.4 Plan

Semester 2

- Week 1: All systems to collect measurements are implemented.

  Deliverable: Latency data is collected via Apache AB, Resource Usage is collected via Prometheus and Grafana.

- Week 2 - 4: Benchmark can run all SMTs on cluster.

  Deliverable: All selected SMTs run the experiment micro-service on cluster

- Week 5 - 6: Automate benchmark

  Deliverable: Benchmark is automated and all SMTs experiments run one after the other without issues

- Week 7 - 8: Run experiments, collect data.

  Deliverable: Use extra time to polish any issues, run experiment and collect data for comparison

- Week 9 - 10: Write-Up

  Deliverable: first draft submitted to supervisor at least two weeks before final deadline.