Comparative Analysis of Service Mesh Platforms in Microservices-Based Benchmark Applications

Francisco Gomes, Paulo Rego, Fernando Trinta, José de Souza

Comparative Analysis of Service Mesh Platforms in Microservices-Based Benchmark Applications

Francisco A. A. Gomes[1], Paulo A. L. Rego[2], Fernando A. M. Trinta[2], and José N. de Souza[2]

[1] Federal University of Ceará (UFC), Crateús, CE, Brazil
almada@crateus.ufc.br
[2] Group of Computer Networks, Software Engineering and Systems (GREat)
Federal University of Ceará (UFC), Fortaleza, CE, Brazil
pauloalr@ufc.br, fernando.trinta@dc.ufc.br, and neuman@ufc.br

Abstract

The development of monolithic systems brings several challenges related to system maintenance and scalability. To mitigate this problem, there is currently a trend in developing systems to use the composition of microservices. The application is divided into independently deployable services, which can quickly implement on any infrastructure resource, and each service runs in an isolated process. Furthermore, the development of technology for microservices also increases the operational complexity associated with modern applications. The service mesh is a promising approach to mitigate this situation, introducing a dedicated infrastructure layer over microservices without imposing modifications to the service implementations. The proposed work aims to present a survey about service mesh platforms and application benchmarks based on microservices and performance tests performed. As a result, tests show that despite the benefits of the service mesh, it impacts application latency.

1 Introduction

The development of monolithic systems brings several challenges related to the maintenance and lack of autonomy of the development teams to adopt new technologies that differ from the adopted architecture [1]. There is currently a trend in systems development to use the composition of microservices, that is, small pieces of cohesive and autonomous software. With the microservice architecture, the application is divided into independently deployable services that can be rapidly developed on any infrastructure resource, and each service runs in an isolated process (container) [3]. Additionally, these services communicate through an HTTP API[1]. Thus, this architectural style allows developers to decompose software into small units, facilitating the scalability of only the most demanded services at that moment, which strongly contrasts with the architectures of corporate systems that are usually a single implementable component.

Several studies point to challenges related to microservices, for example, methodologies and tools for multilayer monitoring, functional adaptation algorithms at runtime, advanced adaptation features at runtime, and high availability support [2]. Furthermore, developing technology for microservices also increases the operational complexity associated with modern applications. This led to the emergence of service mesh, a promising approach to mitigate this situation, which introduces a dedicated infrastructure layer on top of microservices without imposing modifications on service implementations.

[1] Application Programming Interface

The proposed work aims to present a survey about service mesh platforms and its application in a benchmark of applications based on microservices. The benchmark works as a reference to an application that has several microservices related to each other and reflects the idea of a corporate application. Furthermore, the work carried out a performance experiment to analyze the impact of these platforms.

The rest of this article is organized as follows: Section 2 presents the background: microservices, service mesh and platforms. Section 3 presents the results of experiments carried out. Finally, Section 4 concludes the article and exposes possible future works.

2 Background

The microservices emerged empirically from architectural patterns used in the real world, where systems are composed of services that collaborate to achieve their goals, communicating through lightweight mechanisms (e.g., Web APIs) [3]. The idea of microservices is to build small applications, developed independently, which tend to present efficient processing and interoperability aspects, allowing the continuous deployment/delivery of large and complex applications [5]. Given these characteristics, it is possible to implement each microservice with a different technology *stack*. According to Zimnermann (2017) , microservices are a way of implementing and deploying services in SOA using state-of-the-art software engineering practices (i.e., development and deployment paradigms and technologies).

A microservice that communicates with other services incorporates business logic and network communication logic. Each microservice contains a significant part of its network communication-related code, independent of the service's business logic. Implementing functionality related to service-to-service communication from scratch is costly [4]. Instead of focusing on business logic, the developer needs to spend a lot of time creating service-to-service communication functionality. Service mesh is a communication infrastructure between services. With a service mesh, a given microservice will not communicate directly with the other microservices. Instead, all service-to-service communication occurs in a software component called a Proxy sidecar. The sidecar is a software component co-located with the service in the same Virtual Machine or pod (Kubernetes). The Proxy sidecar layer is known as the Data Plane. All these Proxies sidecars are controlled via a control plane. This is where all settings related to inter-service communications apply.

Service mesh provides built-in support for networking functions like resiliency, service discovery, etc. Therefore, service developers can focus more on business logic, while most of the work related to network communication is offloaded to service mesh. The microservice for sidecar communication always takes place via standard protocols such as HTTP1.x/2.x, gRPC, etc. Thus, service mesh is a technology and language-independent. Regarding the control plane, all service mesh sidecars are centrally managed by a control plane. This is useful when supporting service mesh features such as access control, observability, service discovery, etc. All changes made to the control plane are sent to the sidecars.

Several platforms offer *service mesh* for microservice applications. This section will introduce two of them: Istio and Linkerd. Istio[2] is an *open-source* platform for connecting, managing, and securing microservices. It provides an infrastructure for communication between microservices, with resiliency, routing, load balancing, service-to-service authentication, observability, and more, without requiring any changes to the service code. By deploying Sidecar, the developer can add the service to Istio's *service mesh* . Deploying Istio is deeply tied to Kubernetes, but

[2] https://istio.io/

deploying it to other systems is also possible. Linkerd[3] is a *service mesh* for Kubernetes. It makes running services easier and safer by offering run-time debugging, observability, reliability, and security, all without requiring changes to source code. Linkerd is fully open source, licensed under Apache v2, and is a graduate project of the Cloud Native Computing Foundation.

3 Experiments and Results

To verify the use of service mesh platforms and to analyze the impact of these platforms on benchmarks applications in microservices, the work carried out a performance experiment taking into account the latency of a request.

For the experiment, a software called Apache JMeter[4] was used, a tool that allows the performance of load and stress tests in Web applications. The tests were performed on a machine with the following configuration: a node with a 4th generation 1.8 GHz i3 processor, with 2 cores, 8 GB of RAM, 128 GB of SSD, running on the Mint 20 Operating System. In addition, Kubernetes version 1.21 was installed.

For the present research, the test consists of analyzing the Teastore application[5] in three scenarios: with Kubernetes only, Kubernetes with Istio, and Kubernetes with Linkerd. With JMeter, the Teastore application was submit tests considering the return of stored data and calculating the average latency. Each scenario was subjected to different request rates: 600, 1200, and 2400 requests per minute. 10 users were used to running each scenario in concurrency at each request rate.

Figure 1 presents the results of the tests carried out. Baseline is the Kubernetes-only scenario. The obtained results show a better performance of Linkerd compared to Istio in all request rates. When verified at the rate of 2400 requests per minute, the increase in latency was 12.9% comparing the two platforms (Linkerd - 482,2 and Istio - 426,8). Istio even showed better results in some tests than Linkerd, but its overall average was impaired due to the high incidence of errors during the execution of the tests. This can be explained by the fact that Istio has a higher CPU and memory consumption than Linkerd. Another result found is about the impact of service mesh platforms. When comparing Istio with Baseline, the average latency value is up to 6.15 times higher when subjected to a rate of 2400 requests per minute (Baseline - 67,35 and Istio - 482,2).

4 Conclusion and Future Works

The article presented a survey about relevant topics, such as microservices, service mesh and tools, benchmark application, and a performance test performed. Important concepts and definitions for understanding the work were presented, as well as tools that can be used to carry out analyzes in microservices applications. Service mesh brings advantages to the developer because instead of focusing on business logic, it was necessary to spend time creating service-to-service communication functionalities, and with the service mesh is no longer necessary. The results showed that despite the benefits of service mesh, it impacts the latency of applications. The conclusion is that, in terms of performance, Linkerd is better than Istio, which is explained by its lighter system, that consumes fewer resources. However, there are still reasons why a developer might prefer to use Istio, more popularity and greater community support.

[3] https://linkerd.io/
[4] https://jmeter.apache.org/
[5] https://github.com/DescartesResearch/TeaStore

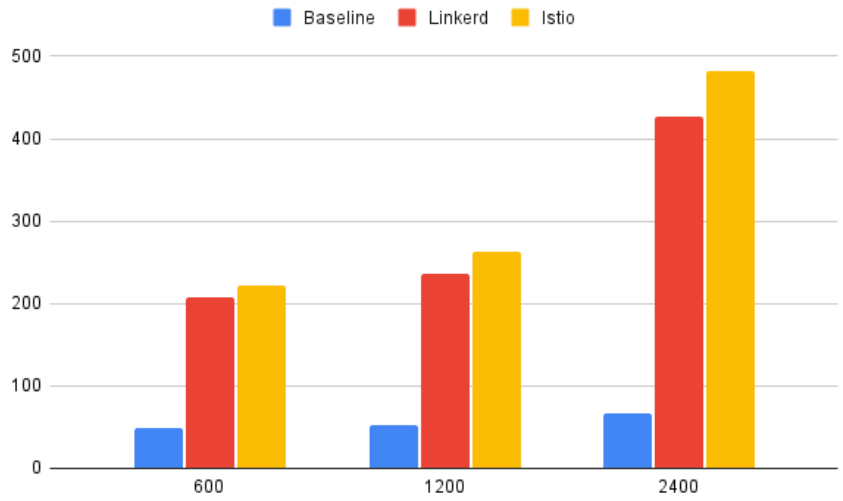

Figure 1: Comparative Analysis of Tests

As future work to improve this research, it is intended to carry out:

- Tests in other benchmarks in order to verify if the same behavior remains in relation Kubernetes, Linkerd, and Istio;
- Use another Service Mesh platform for analysis in performance tests; and
- Perform more performance tests and analyze other metrics (CPU, RAM, Disk)

References

[1] Keith H. Bennett and Václav T. Rajlich. Software maintenance and evolution: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, page 73–87, New York, NY, USA, 2000. Association for Computing Machinery.

[2] Emiliano Casalicchio and Stefano Iannucci. The state-of-the-art in container technologies: Application, orchestration and security. *Concurrency and Computation: Practice and Experience*, 32(17):e5668, 2020. e5668 cpe.5668.

[3] Martin Fowler and James Lewis. Microservices. 2014.

[4] Kasun Indrasiri and Prabath Siriwardena. Service mesh. In *Microservices for the Enterprise*, pages 263–292. Springer, 2018.

[5] Sam Newman. *Building microservices: designing fine-grained systems.* " O'Reilly Media, Inc.", 2015.

[6] Olaf Zimmermann. Microservices tenets. *Computer Science-Research and Development*, 32(3-4):301–310, 2017.