Research on Enterprise Service Governance Based on Service Mesh

Yuchuan Tian[1], Haitian Hao[2], Xiaojian Xu[3] and Bing Liang[4]

[1] School of Electrical and Electronic Engineering, North China Electric Power University, Beijing, 102206, China
[2] Department of Computer Science, University of Maryland, College Park, MD 20742, US
[3] Software Development Center, China CITIC Bank, Beijing, 100010, China
[4] STATE GRID Beijing Changping Electric Power Company, Changping district, Beijing, 102200, China.
Email: ohao@outlook.com

Abstract. With the technology development, enterprise applications are rapidly changing from the traditional centralized application architecture to the distributed, microservices architecture (MSA). However, with the further development of MSA, enterprises are facing new challenges while enjoying the convenience brought by microservices, e.g. how to manage a large number of services, how to quickly locate and troubleshoot these services, how to avoid the performance issues, etc. As the next generation of MSA, service mesh is attracting more and more attentions and can be used to solve above problems. On top of service mesh, this paper proposes one mechanism that largely customizes service mesh implementation technologies. Furthermore, we implement this mechanism in a real project of one local large financial company. The final result illustrates that the mechanism is not only feasible but also effective to solve aforementioned service governance problems.

1. Introduction

Following the evolution of the cloud computing paradigm in the past decades, enterprise modern applications are more commonly to be designed with microservices architecture (MSA) [1]. Supported by the technologies such as Docker [2], Kubernetes (K8S) [3], and Jenkins [4], the methodologies of MSA dramatically reduces the incremental operational burden to software service deployment. However, despite significant improvement in both efficiency and productivity during the development and deployment phases, the operational complexity during service runtime has not been mitigated. To mitigate this situation, a promising approach Service Mesh [5] has emerged in recent years which introduces a dedicated infrastructure layer over microservices without imposing modification on the service implementation.

Currently modern enterprise applications consist of a large number of microservices which might be implemented using different languages, may belong to different tenants, and have thousands of services instances with constantly changing states that are caused by the platform scheduling. Although some service mesh platforms, including Istio [6], Linkerd [7], etc., can provide the fundamental features for service governance, enterprises still need to tackle the major challenges themselves like high performance, high adaptability, debugging [8], high availability [9], etc.

This paper proposes one practical approach Enterprise Service Mesh Platform (ESMP) to solve the aforementioned problems by modifying Envoy [10] and changing other components, e.g. Mixer [11]

to avoid potential performance bottleneck and other issues. A real project that implements this approach demonstrates the feasibility and effectiveness.

2. Issue Discussions and Design Principles

2.1. Issue Discussions

Although there are several service mesh platforms in the industry, they are not mature enough for enterprise production level usage due to below reasons:

2.1.1. Cannot support all requested communication protocols. Enterprise is utilizing many protocols in existing applications, such as HTTP/rest, Dubbo RPC, web service, socket, etc. Unfortunately none above platforms can support all these protocols well.

2.1.2. Direction of sidecar technology. As Istio's sidecar proxy, Envoy is implemented with C++, which is not often used in enterprise. Enterprise is now widely using Golang instead, which has similar performance as C++. Considering this direction, the sidecar will need to be revised.

2.1.3. Performance issue. One Istio component, mixer has always been challenged about the performance issue, and the issue is even more critical for financial companies with large-scale cluster deployments. To mitigate this situation, we propose one Enterprise Service Mesh Platform (ESMP) that can be implemented in enterprise production environment by largely revising and customizing the components based on Istio.

2.2. Design Principles

In general, service mesh is designed to provide a set of fundamental features, including service discovery, load balancing, fault tolerance, traffic monitoring, circuit breaking, authentication and access control, etc. When migrating to service mesh architecture, one enterprise needs to fully consider all factors including current technology stack, existing system architecture, strategic plan and the business demands etc. The following principles should be well defined and well followed:

2.2.1. Principle one: Minimum cost. The Service governance should be fulfilled without any invasion of the business system logic.

2.2.2. Principle Two: The new platform should be compatible with the mainstream technology stack. Enterprise applications are implemented with different languages, mainly Java, using wide range of architectures, e.g. Dubbo, Spring Cloud, web service, socket etc.

2.2.3. Principle Three: High adaptability. The platform should support multiple deployment architectures. Enterprise has a variety of deployment architectures, including VM, private cloud and public cloud etc. And there are cross calls among various deployment environment services.

2.2.4. Principle Four: High availability. There must be no business interruption. Any change or upgrade cannot affect the availability of the business system.

3. Design and Detailed Solutions

3.1. Overall Design

As ESMP is built based on the enterprise PaaS cloud platform that supports container environment, PaaS platform will ensure the high availability of ESMP by providing necessary hardware and software support. We only focus on the following components from functional perspective:

3.1.1. Mesh-mgmt: platform management console. It provides a visual management interface for policy configuration and routine management functions of service governance, such as traffic control

strategy, service call topology, health detection, etc., and obtains PaaS platform service capabilities, such as monitoring information, log query, etc., by calling PaaS platform API interface.

3.1.2. Mesh-proxy: mesh platform agent. Encapsulating the K8S APIs, it provides external API functions of platform to access related resource objects on K8S by calling K8S API server, and other services on container cluster in the form of proxy, such as Nacos, Jaeger, etc.

3.1.3. Mesh-pilot: the control plane of Service Mesh. It gets user-defined rules from K8S API (defined by CRD), and sends configuration information to sidecar with the format of XDS protocol.

3.1.4. Nacos: unified registration center. As the registration center of Dubbo and Spring Cloud etc., it allows user services to register through the SDK and manage the metadata that the SDK relies on. With modification of Nacos, the final data will be stored in etcd through K8S-apiserver. After a service registers with Nacos, a correspondent K8S service with an assigned cluster IP will be created to ensure the consistency.

3.1.5. Mesh-sidecar: data plane of Service Mesh. Developed with Golang, it will replace Istio's original sidecar envoy. It will intercept all the incoming and outgoing traffics of services, and execute traffic limiting, fusing, degradation, authentication and other functions according to the distributed configuration rules.

3.1.6. Jaeger: component to trace the call chain. It integrates all call information among services, generates service call chain data, and provides external query API.

3.1.7. Mesh-sdk: client sdk. It is used by Dubbo, Spring Cloud services to register with Nacos, discover service and get call chain information reporting, etc.

3.1.8. Mesh-vmctl. These are a set of packages including components like calico_node, dnsmasq, etc. are responsible for connecting services on the virtual machine (VM) to the Service Mesh, like network configuration, deployment of sidecar related components, start and stop of services on the VM, etc. The overall architecture design is shown as figure 1.

3.2. Solution for Traditional Architecture Support

The best practice of service mesh is to implement in cloud native application, unfortunately many enterprise applications are still built with traditional architecture, e.g. centralized single machine or VM environment. As these legacy systems are still supporting one enterprise's main business operations, especially for financial companies, how to integrate these applications into ESMP is the first problem in the successful implementation.

ESMP is built based on PaaS container environment, and implemented the communications among VM and container applications. There are two main scenarios:

- If VM and K8S cluster are in the same subnet, then calico mode can be set to BGP, the data is routed to POD via K8S node load;
- If VM and K8S cluster are not in the same subnet, VM and K8S nodes need to be routed through routers. Then calico mode needs to be set to IPIP, and a connection channel tunnel is established between VM and K8S container network. Source IP and destination IP are packaged and unpacked at both ends of the channel, so as to realize mutual discovery between routes across subnets. In the end, a virtual network interface will be added to the routing table of each node in the container cluster.

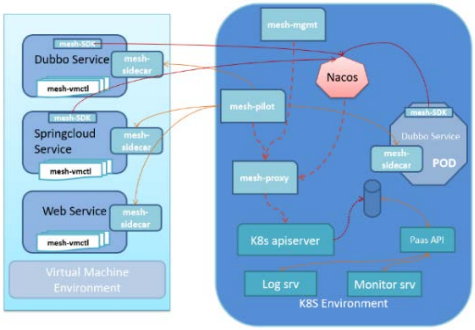

Figure 1. Overall architecture design.

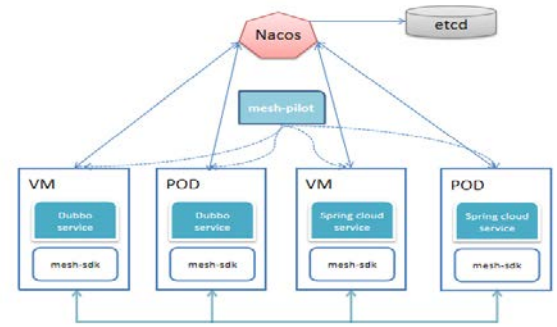

Figure 2. Service link topology.

3.3. Solution for Service Link Topology

Modern enterprises have large number of microservices and the call relationships among these microservices should be recorded and tracked. ESMP uses the integrated call chain tracing component Jaeger to generate service call chain information. In order to support the collection of call chain information of Dubbo service, the platform uses Zipkin client to replace Jaeger client. Both Zipkin and Jaeger comply with the OpenTracing standard, and Zipkin client can set the sampling rate according to the actual traffic situation of the service. To avoid performance impact, the sampling rate of the service with less traffic can be set higher, or even 100%, while for the service of more traffic business system, the sampling rate needs to be set lower. The call chain information is finally collected by Jaeger collector and stored in elasticsearch, as shown in figure 2.

3.4. Solution for Unified Registration

Enterprise applications with MSA architecture mainly registers and discovers the services through the registration center, like zookeeper, Eureka and Nacos etc. This will cause the waste of resources, and also make it difficult to manage the services. Considering the compatibility with Dubbo and Spring Cloud, we choose Nacos as the unified registration center.

ESMP integrates Nacos and K8S to realize unified registration of multiple protocols. Dubbo and Spring Cloud services with MSA architecture can register services to Nacos via referencing SDK. Nacos can call K8S API to save the data in etcd to achieve data consistency. As Nacos can support various protocol frameworks, the metadata will be processed in a unified way to achieve the unification of service registration data model, refer to figure 3.

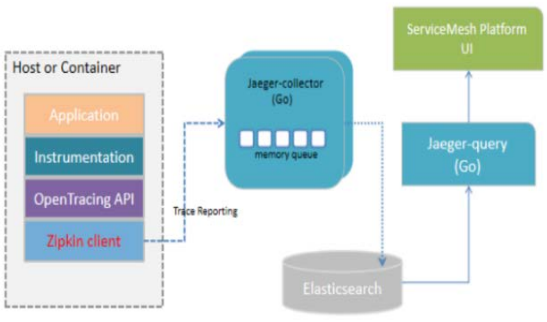

Figure 3. Unified registration center.

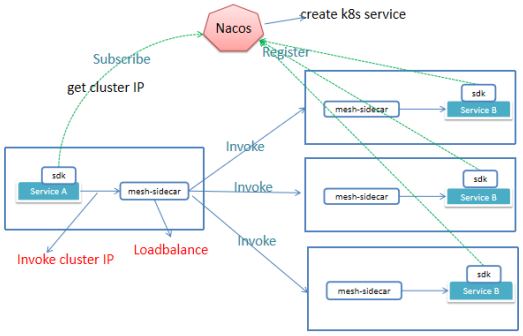

Figure 4. Load balance.

3.5. Solution for Load Balance Conflicts

Another step for ESMP is to block the load balance function of the original microservices client and change to use the function provided by sidecar so as to avoid the load balance conflicts.

ESMP uses a unified SDK for the services of Dubbo, Spring Cloud and other MSA frameworks. The service address discovered by modified Nacos service is K8S's ClusterIP, enabling MSA clients to

make service calls using ClusterIP. Once a service call is made through ClusterIP, traffic is intercepted by sidecars. Finally the load balancing policy will be executed according to the configuration dispatched from mesh-pilot to the sidecars, refer to figure 4.

3.6. Solution for traffic Control

An important function of ESMP is traffic hijacking and control, which makes the incoming and outgoing requests of the business system go through sidecar first so that sidecar can hijack and control the traffic.

In original Istio, one platform needs to remotely check with Mixer to realize the traffic limitation. ESMP, on the other hand, doesn't rely on the mixer. To achieve the goal of high performance, it directly makes judgments with the sidecar instead. Two algorithms are supported, i.e. QPS and rate limit. For HTTP requests, traffic limitation can be judged by matching version, header, path information, etc. While for Dubbo requests, this can be done by matching version, header, etc.

4. Case Study

The ESMP has been successfully implemented in one large domestic financial company and the final results are quite encouraging. ESMP can completely support all necessary service governance features that establish solid basis for future migration, and meanwhile achieve many business benefits. Firstly, after decoupling service governance and business logic, several public services such as traffic control, service authentication, intelligent routing, unified registration etc. can be sunk into infrastructure layer, which can save 20-30% of the development cost. Secondly, through health monitoring, state early warning, fault simulation and other functions, ESMP can realize rapid operation and maintenance. The efficiency of fault location and troubleshooting has been increased by 90%. Finally, through the visual operation interface, it is easy to complete the configuration and distribution of various strategies. And with the help of the service call topology map and service list, one can easily tell the complex service call, strong and weak dependence within the enterprise.

Below are the detailed introductions of the two most significant benefits, the rapid operation and maintenance of the system through global service call link topology diagram and the prevention of the occurrence of system avalanche through traffic control.

4.1. Global Service Call Link Topology

When any issues or incidents happen, it is critical for the engineers to tell which microservices nodes work abnormally. Unfortunately, as enterprise has large amount of microservices, it is a challenge to master and trace the call relationships among microservices timely.

Once any issue happens on the front-end, the engineers have to spend several hours or even longer to look into all log information to get services details one by one, e.g. recent calls among microservices, call details like success, failure, response time, etc.

After implementing ESMP, one can easily trace the service calls and status via service call link topology graph at any time, as shown in figure 5.

4.2. Utilize Traffic Control to Prevent System Avalanche

Normally enterprises will carry out some business promotion events every year. And there will be up to hundreds of times growth in concurrent user numbers at peak time, e.g. at the beginning of the promotion event, and this will introduce a peak flow and will seriously influence the performance of relevant applications.

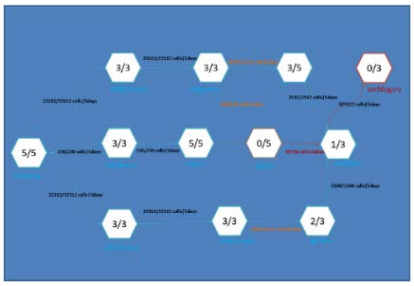

Figure 5. Global Service Call Link Topology.

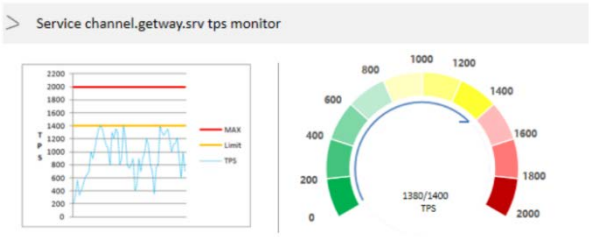

Figure 6. Traffic control to prevent system avalanche.

As relevant applications have various processing capacity, ranging from 5000TPS to a few hundred TPS or even less. If the front-end system has much higher capacity and the back-end side cannot process timely, this will cause a series of disasters without traffic control. The back-end system will crush, and this will cause a chain reaction and finally lead to an avalanche of all relevant systems. With the help of ESMP's traffic control function, the traffic flow of the front-end system will be limited within the processing capacity of the back-end system, and thus ensure the overall stability of all the business systems, as shown in figure 6.

5. Conclusion

Based on a real case, this paper puts forward a solution of service governance for enterprise applications based on service mesh technology. The scenarios, issues and solutions described in this paper are actually very typical in all enterprises, especially for the financial companies. ESMP can not only provide enterprises with convenient service governance functions, reduce business system development costs, improve operation and maintenance efficiency, but also provide standards and specifications for enterprise IT governance.

There are still a lot of rooms for the development of service mesh technologies. This paper studies and implements the service mesh technologies for specific business scenarios, and further studies are still needed for other scenarios, e.g. general integration scheme of service mesh technology.

6. References

[1] Balalaie A, Heydarnoori A, and Jamshidi P 2016 Microservices architecture enables DevOps: migration to a cloud-native architecture *IEEE Software*, **33** 42–52

[2] Docker Inc 2020 Docker: enterprise container platform https://www.docker.com/

[3] The Cloud Native Computing Foundation 2020 Kubernetes: ProductionGrade Container Orchestration https://kubernetes.io/

[4] The Jenkins Project 2020 Jenkins https://jenkins-ci.org/

[5] Buoyant Inc 2017 A Service Mesh for Kubernetes https://dzone.com/articles/whats-a-service-mesh-and-why-do-i-need-one

[6] Zhou X, Peng X, Xie T, Sun J, Li W, Ji C, and Ding D 2018 Delta debugging microservice systems *Proc. of the 33rd ACM/IEEE Int. Conf. on Automated Software Engineering (ASE '18) ACM* 802–7.

[7] Sheikh O, Dikaleh S, Mistry D, Pape D, and Felix C 2018 Modernize digital applications with microservices management using the Istio Service Mesh *Proc. of the 28th Annual Int. Conf. on Computer Science and Software Engineering (CASCON '18). IBM Corp.* 359–60.

[8] The Istio Team 2020 Istio: an open platform to connect, manage, and secure microservices https://github.com/istio/istio

[9] The Cloud Native Computing Foundation 2020 Linkerd: production-grade feature-rich Service Mesh for any platform https://github.com/linkerd/linkerd

[10] The Cloud Native Computing Foundation 2020 Envoy https://github.com/envoyproxy/envoy

[11] The Istio Team 2020 Mixer https://github.com/istio/istio/tree/master/mixer