Service Mesh: Challenges, State of the Art, and Future Research Opportunities

Wubin Li and Yves Lemieux
Ericsson Research, Ericsson, Montréal, Canada
{wubin.li, yves.lemieux}@ericsson.com

***Abstract*—While the technology development towards microservices can significantly improve the speed and agility of software service delivery, it also raises the operational complexity associated with modern applications. This has led to the emergence of *Service Mesh*, a promising approach to mitigate this situation by introducing a dedicated infrastructure layer over microservices without imposing modification on the service implementations. Aiming to inspire more practical research work in this exploited area, we in this paper present a comprehensive review on the state of the art of Service Mesh and discuss the related challenges and its adoption. Finally, we highlight the opportunities for future research in this subject.**

***Index Terms*—Microservices, Service Mesh, Service Oriented Architecture, DevOps, Service Delivery, Challenges, Opportunities, Edge Computing**

I. Introduction

Following the evolution of the cloud computing paradigm in the past decades, there also has been a few significant shifts in fundamental abstraction within the software service delivery domain. As presented in Table I, for example, rather than *on-premises* design, modern applications are more commonly to be designed with cloud-native [1] architectures, enabling executions in orchestrated environments. One major outcome of these shifts is the trend of *DevOps* [2], which has already become mainstream in the enterprise [3]. Its coalescence of development and operations can be attributed to the recent technology development towards microservices [1] which effectively standardize the patterns for software packaging and deployment, and thus significantly improve the speed and agility of software service delivery.

The core proposition of microservices is to break down a complex monolith into small services that can be independently deployed and maintained. Driven by advantages of microservices, e.g., flexibility, simplicity, and scalability, there has been a massive adoption of microservices in different areas, e.g., for big data processing [4], [5], for IoT applications [6], [7], and for high performance computing (HPC) [8], [9]. Advocates of microservices have come to believe that this can not only solve enterprise application integration problems but also simplify the plumbing required to build service-oriented architectures (SOA).

Indeed, backed by DevOps-supporting technologies such as Docker [10], Kubernetes [11], and Jenkins [12], the methodologies of microservices dramatically reduces the incremental operational burden to software service deployment. However, despite significant improvements in both efficiency and productivity during the development and deployment phases, the operational complexity during service runtime has not been mitigated as a result.

In particular, a cloud-native application might consist of a large number of microservices which (I.) might be implemented using different languages, (II.) might belong to different tenants, and (III.) might have thousands of service instances with constantly changing states that are caused by the orchestration platform scheduling. In such dynamic environments, debugging microservices applications turns out to be a challenging task [13] due to the complexity of service dependencies and the fact that any service can become temporarily inaccessible to its consumers [14]. Additionally, as the application behavior depends on the flow of the traffic among microservices, traffic control becomes a must (but currently missing) for runtime operation.

This has led to the emergence of *Service Mesh*, a promising approach to mitigate this situation by introducing a dedicated infrastructure layer over microservices without imposing modification on the service implementations. Serving as a fully manageable service-to-service communication platform, a service mesh is designed to standardize the runtime operations of applications. As part of the microservices ecosystem, the service mesh technologies hold the promise in addressing communication-related concerns [15], e.g., interoperability [16], traffic segmentation [17], dependency control [18], runtime enforcement, etc.

Our main contribution in this paper is a comprehensive study on the challenges, state of the art, and future research opportunities in regards to service mesh. To the best of our knowledge, this is the first paper to systematically review the literature of service mesh and the key technologies behind the theme. Given the current state of the research on service mesh and the cloud-native evolution, we believe our effort is just-in-time. We expect that it can attract valuable research interest and inspire more practical research work in this exploited area.

The remainder of this paper is organized as follows. In Section II, we elaborate the definition of service mesh as well as its fundamental features. Section III surveys the challenges of service mesh, as well as the state of the art. Section IV presents future research opportunities in the studied area. The

TABLE I. Selected categories of fundamental abstractions changed in the past decades.

Category	The Past	The Present
Infrastructure & Execution Environment	Data Centers	Orchestrated Environments
	Bare Metal / Virtual Machines	Containers
Network Concerns	IP Address, DNS	Service Discovery
	TCP/IP	gRPC, REST, ...
Software Design	Monolithic Applications	Microservices
	Hardware Redundancy	Design for Failure
	On Premises	Cloud Native
Development Methodology	Isolation between Development and Operations	DevOps

paper is concluded in Section V followed by acknowledgments and a list of references.

II. Service Mesh

A. The Definition of Service Mesh

Though controversial, we believe that the concept of Service Mesh was first articulated by William Morgan from industry. A detailed definition [19] of service mesh can be presented as the following:

A service mesh is a dedicated infrastructure layer for handling service-to-service communication. It's responsible for the reliable delivery of requests through the complex topology of services that comprise a modern, cloud native application. In practice, the service mesh is typically implemented as an array of lightweight network proxies that are deployed alongside application code, without the application needing to be aware.

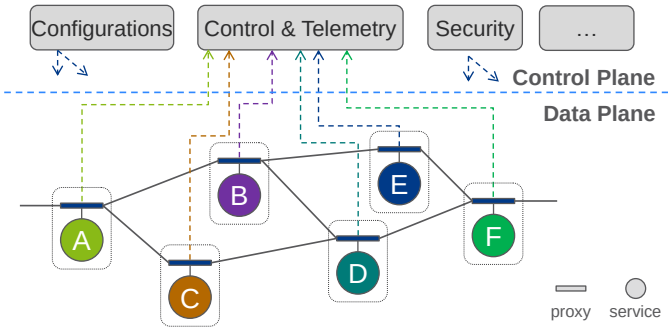

Fig. 1: An architectural overview example of the Service Mesh approach.

The definition is quite straight-forward. It can also be illustrated through the example presented in Figure 1. As shown, a service mesh infrastructure logically consists of a data plane and a control plane. The data plane is composed of a set of intelligent proxies, which are typically deployed as sidecars [20]. These proxies mediate and control all network communication between microservices. Having visibility to each network packet, the main responsibilities of the data plane include service discovery, health checking, routing, load balancing, authentication/authorization, and observability. The control plane manages and configures the proxies for traffic routing. Additionally, the control plane configures corresponding components to enforce policies and collect telemetry. The control plane works as the brain of a service mesh. It does not require the visibility into the network traffic.

B. The Fundamental Features

In general, a service mesh is designed to provide a set of fundamental features, as described in the followings.

- Service discovery: In microservices applications, the number of service instances as well as the states and location of a service are changing dynamically over time. Thus, a mechanism of service discovery is required to enable service consumers to discover the location and make requests to a dynamically changing set of ephemeral service instances. Typically, service instances are discovered by looking up a registry underneath which keeps records of new instances as well as instances that are removed from the network. This is critical feature of a service mesh.
- Load balancing: Supported by the service discovery, the load-balancer in a service mesh provides the capability of traffic routing across the network. Compared to simple routing mechanisms (e.g., round robin, and random routing), modern service mesh load-balancing routing now can consider latency and the state (e.g., health status, and current variable load) of the backend instances.
- Fault tolerance: From a networking model perspective, a service mesh sits at a layer of abstraction above TCP/IP. With the assumption that the network underlying L3/L4 network, as with every other aspect of the environment, is unreliable, the service mesh must therefore also be capable of handling failures. This is typically achieved by redirecting the consumer requests to a service instance with healthy state.
- Traffic monitoring: All communication among microservices are to be captured, enabling reporting of requests volume per target, latency metrics, success and error rates, etc.
- Circuit breaking: In case of accessing an overloaded service that has high latency already, the capability of circuit breaking will automatically back off the requests rather than completely failing the service with excessive load and resulting in propagated unavailability.
- Authentication and access control: Through policy enforcement from the control plane, a service mesh can define which services are allowed to be accessed by which services, and what type of traffic is unauthorized and should be denied.

With these fundamental features, complex functionalities can be built, e.g., through authentication and access control, an identity-based routing policy can be implemented and enforced.

III. Challenges and the State of the Art

A. Three Main Challenges

To achieve the vision of a service mesh, three main challenges can be identified as the following:

I. The first is the design supporting high performance. At the data plane level, the proxy component co-located with the service instances is the heart of a service mesh. It is responsible for intercepting and mediating the traffic among microservices and needs to be light-weight and designed for high performance. The same requirement applies to the components in the control plane, especially for those responsible for data aggregation.

II. The second is adaptability. To support and adapt to a wide range of cloud-native orchestration platforms, a service mesh needs to provide a certain level of configurability, extensibility, and pluggability.

III. Third, high availability is also a very critical concern. A highly available service mesh can lower the risk of decision-making with bias due to data/component unavailability. As discussed in Section IV-A, it is also crucial to isolate the impact of unavailability on the application side from service mesh.

Some of these challenges are being addressed by current solutions, e.g., for performance, the data plane proxy in Linkerd [21] is packaged as a image with size less than 10 mb and normally can be up and running within 1 ms. Meanwhile, these challenges also present opportunities for future research as will be elaborated in Section IV.

B. Enablement Technologies

Table II presents four service mesh platforms which we believe are the most attractive ones, including Istio [22], Linkerd [21], Amazon App Mesh [23], and Airbnb Synapse [24].

Specifically, Istio and Linkerd both provide all the fundamental features highlighted in Section II-B. While Istio has a more active community that runs the project with faster iteration cycles, Linkerd is more stable in our hands-on experimental study and we believe it is production-ready. Another important fact is that Linkerd is a CNCF[1]-accepted project, while Istio is still on its way to CNCF. On the other hand, Istio provides a large number of flexible APIs for developers, allowing researchers to efficiently experiment with new ideas.

Unlike Istio and Linkerd, Airbnb Synapse does not have an abundant feature set. Instead, it solves the problem of automated failover through a simplistic but efficient, Zookeeper-based service discovery mechanism. In detail, Synapse configures a local HAProxy process using the information read from Zookeeper where states of the services are stored and continuously updated. By doing so, the local HAProxy can take care of properly routing the request from a service consumer. With an optimized HAProxy, Synapse can react to the change (e.g., service failure) in Zookeeper, and reconfigure HAProxy immediately. Synapse has been proven to be with broad applicability, however, it was not originally designed with the vision of service mesh. And thus, the project is becoming less active as it matures.

Recently (On 28, Nov 2018), Amazon, the largest cloud vendor, also announced AWS App Mesh, a service mesh solution for microservices on AWS. The best and worst part of AWS App Mesh is its native compatibility with Amazon Web Services. This might guarantee a minimum integration effort from users and probably provides best-effort performance, but at the same time, it might also result in vendor lock-in issues and inapplicability to hybrid cloud environments.

Finally, please note that there are other service mesh platforms which are built based on the aforementioned. For example, SOFAMesh [25] replaces Envoy [26] with MOSN [27] (written in GoLang) for further performance improvements at data plane. It also makes modification to other components, e.g., moving Mixer [28] from the control plan to the data plane, in order to avoid potential performance bottleneck. However, we still consider it as an variation of Istio. Another remarkable point is that the aforementioned solutions lack the support for distributed cloud scenarios as they are mainly designed for centralized (single-cloud) environments.

C. An Overview of Relevant Research Literature

One surprising finding of our research is that there are very few research work on service mesh from academia. We believe the main reason is that the concept of service mesh is new, which also motivates this survey paper. Table III briefly summarizes a set of publications with service mesh mentioned. As presented, none of these dives deeply into service mesh, but instead, they are focused on complementing existing/proposed solutions using service mesh.

IV. Future Research Opportunities

Although service mesh is still in its infancy, it already significantly complements microservices architectures in terms of observability, traceability, and manageability. Meanwhile, we believe that it also opens a few opportunities for future research.

A. Highly Available Service Mesh

The service mesh layer provides uniform, global mechanisms to both control and measure all request traffic between microservices. At the data plane level, this is implemented by providing a light-weight proxy for each service instance. Although modern container-orchestration platforms (e.g., Kubernetes, Mesos [32]) are designed to seamlessly handle service failures, the service availability might still possibly be harmed by the failure introduced by the proxy. The same situation

[1]CNCF, namely the Cloud Native Computing Foundation, is an open source software foundation that hosts and nurtures cloud-native projects like Kubernetes, and Prometheus, aiming at making cloud native computing universal and sustainable.

TABLE II. An comparison of existing service mesh solutions (selected).

Platform	Data Plane	Open Source	Activeness	Major Advantage	Critical Limitation	Overall Maturity
Istio	Envoy	Yes	Good	Growing Community & Fast Iteration	Lack of Support	Moderate
Linkerd2	Linkerd-proxy	Yes	Good	Stability & CNCF Accepted	Potential Vendor Lock-in	Good
AWS App Mesh	Envoy	No	Good	Native Compatibility with AWS	Closed Ecosystems	In Preview
Airbnb Synapse	HAProxy/Nginx	Yes	Poor	N/A	Very Limited Features	Poor

TABLE III. Academia Publications on Service Mesh.

Publications	Year	Focus
[20]	2018	Promoting Service Mesh
[29]	2018	Microservices Governance
[16]	2018	Using Service Mesh to Support IoT Cloud Application
[30]	2018	Architectural Solution for Edge Environments
[15]	2018	Service Mesh supported MAPE [31] loop control

can be applied to the components on the control plane, e.g., if the component that is responsible for authentication and authorization is temporally unavailable, the behavior of the application may result in an inconsistent state where services may receive requests that are supposed to be tagged as invalid. In particular, Istio fails to meet the requirements of availability and performance by introducing a critical dependency in the request path [33].

Therefore, a highly available service mesh is not only crucial to guarantee its promised functionalities, but also to ensure the availability of the proxied services is not affected. However, at the time of this writing, this challenge is still open in existing solutions.

B. Analytics on top of Service Mesh

We also observe the substantial opportunity for data analytics on microservices applications presented by the service mesh layer. Through the proxies that intercept all network communication between microservices, the service mesh layer is able to collect the core monitoring data in software systems, i.e., metrics, logging, and traces, as identified by Peter Bourgon [34].

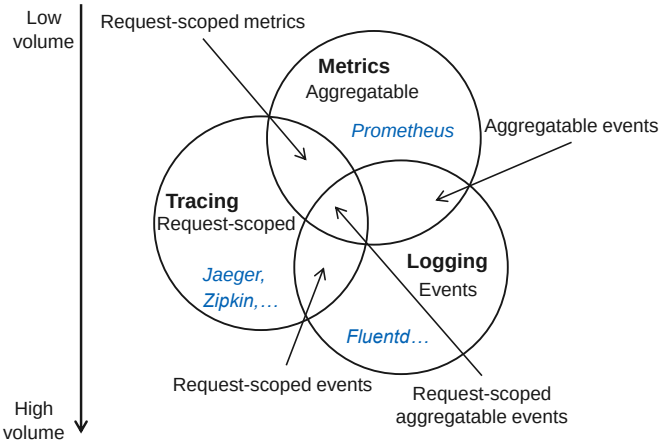

Fig. 2: Venn diagram for monitoring data in cloud-native applications. Source from [34] with minor modification.

These three categories of data can be illustrated in Figure 2. Current existing service mesh solutions have extensive integration support respectively for each category, e.g., Prometheus [35] for metrics which are aggregatable, Jaeger [36] and Zipkin [37] for distributed tracing, and Fluentd [38] for event logging. This allows researchers and developers to minimize the effort of building the pipelines for data collection and focus on data analytics. Depending on the use case context, example could be event mining, anomaly detection, and service dependency extraction, etc.

C. Enhancement on Dependency Management, and Performance Tracing and Analysis

One attractive scenario of using service mesh is dependency management [18], and performance tracing and analysis for microservices applications. Existing extension supports, e.g., Jaeger and Zipkin as mentioned in the previous subsection, are intrusive as they are based on the OpenTracing [39] framework which requires instrumentation to the applications using OpenTracing API specification. While this approach can help precisely pinpoint where failures occur and what causes poor performance, such instrumentation is not always realistic, e.g., when service implementations are not exposed (i.e., black-box services), and when a container image (with encapsulation of service implementation) is not allowed to be modified due to copyright/license issues, etc.

Hence, an enhancement with non-intrusive mechanisms becomes a necessity to expand the use case scenarios on performance tracing and analysis. One promising direction would be to investigate the feasibility of integrating inference-based approaches, such as Project5 [40], WAP5 [41], Microsoft's Sherlock [42], and Facebook's The Mystery Machine [43]. In order to adapt the application context of service mesh, we expect this may require extensive effort with both research and engineering.

D. Service Mesh in Edge Computing Environments

As presented in Section III-B, existing service mesh solutions are mainly designed for centralized cloud environments which are typically located within a single data center, and thus have very limited support for distributed cloud scenarios. This situation is exaggerated for the emerging edge cloud paradigm [44] which advocates to distribute resources along the path between centralized data centers and the logical endpoints of a network (that may have an increasingly large number of devices) in order to improve the performance, operating cost and reliability of applications and services. On the other hand, the rising adoption of container technologies [45], [46] and microservices methodology [47], [48] in edge computing [49], has indicated the need and trend of exploration of

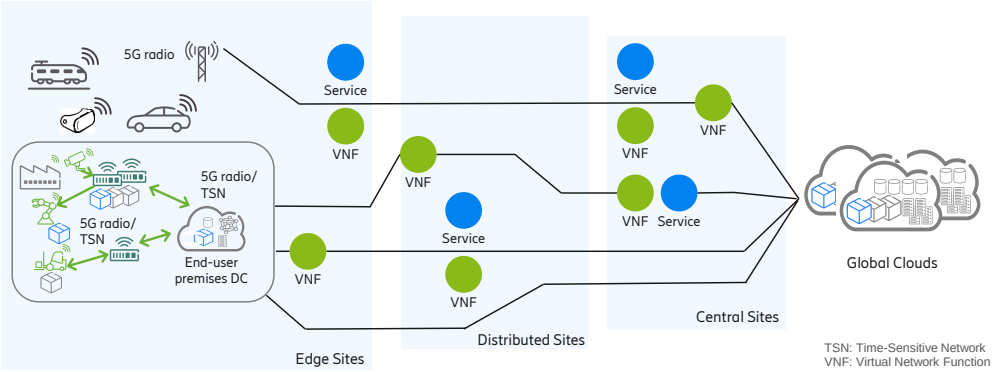

Fig. 3: A mobile edge cloud deployment with 5G.

service mesh in this area as the merging applications running on edge clouds are expected to be designed cloud-natively.

From the perspective of telecommunication industry, it would be even more interesting and challenging to explore how service mesh can be applied in Mobile Edge Computing (MEC [50]) environments. The mobile edge cloud, backed by 5G technologies, is designed to support services requiring low latency or massive data offload, e.g., intelligent transport systems, cooperative control, and VR/AR applications.

As illustrated in Figure 3, a deployment of a mobile edge cloud may consist of multiple sites, which usually are heterogeneous in their HW/SW setup, geographically distributed, and typically structured hierarchically. Besides, multiple VNFs[2] (Virtual Network Functions [51], which may belong to different tenants) reside within the same sites where the services are provisioned. In consequence, such complexities will result in extensive effort to identify the problems and issues when migrating the service mesh into the mobile edge cloud, and design the required extensions on top of existing service mesh solutions in order to tackle the identified problems and make them optimized for mobile edge cloud environments. Example of such extensions could be optimized load balancing or service routing mechanism, with support for multi-tenancy, multi-protocol, and flexible networking typologies.

V. Concluding Remarks

Microservices has already changed the patterns on application design and deployment, but along with them brought runtime operational challenges including service discovery, traffic routing, failure handling, and visibility to microservices. In this survey paper, we systematically reviewed the state-of-the-art on Service Mesh, an emerging technology to reduce the operational complexity for microservices applications. We presented the challenges and opportunities in the related fields, and discussed a few valuable topics for future research, especially in telecommunication industry. We believe that as the evolution of cloud-native technologies and microservices, academic research and industrial development on Service Mesh will grow substantially.

[2]VNFs, aka., virtualized network functions, are virtualized tasks formerly carried out by proprietary, dedicated hardware but now moved into software that runs on commodity hardware.

Acknowledgments

We would like to thank the anonymous reviewers for their constructive feedback. We are also grateful to Xuejun Cai and Catherine Truchan for their insightful advice on improving the quality of this paper. This work is financially supported by Ericsson Research, Montréal, Canada.

References

[1] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices Architecture Enables DevOps: Migration to a Cloud-native Architecture," *IEEE Software*, vol. 33, no. 3, pp. 42–52, 2016.

[2] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional, 2015.

[3] Dimensional Research and LightStep, "Global Microservices Trends: A Survey of Development Professionals," April 2018. [Online]. Available: https://go.lightstep.com/global-microservices-trends-report-2018

[4] J. Gao, W. Li, Z. Zhao, and Y. Han, "Provisioning Big Data Applications as Services on Containerized Cloud: A Microservices-based Approach," *International Journal of Services Technology and Management*, 2019, to appear.

[5] J. L. Schnase, D. Q. Duffy, G. S. Tamkin, D. Nadeau, J. H. Thompson, C. M. Grieg, M. A. McInerney, and W. P. Webster, "MERRA Analytic Services: Meeting the Big Data Challenges of Climate Science through Cloud-enabled Climate Analytics-as-a-service," *Computers, Environment and Urban Systems*, vol. 61, pp. 198–211, 2017.

[6] A. Krylovskiy, M. Jahn, and E. Patti, "Designing a Smart City Internet of Things Platform with Microservice Architecture," in *Proceedings of 3rd International Conference on Future Internet of Things and Cloud*, ser. FiCloud '15. IEEE, 2015, pp. 25–30.

[7] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic Computing: A new Paradigm for Edge/Cloud Integration," *IEEE Cloud Computing*, vol. 3, no. 6, pp. 76–83, 2016.

[8] F. Z. Benchara, M. Youssfi, O. Bouattane, and H. Ouajji, "A new Efficient Distributed Computing Middleware based on Cloud Microservices for HPC," in *Proceedings of 5th International Conference on Multimedia Computing and Systems*, ser. ICMCS '16. IEEE, 2016, pp. 354–359.

[9] C. de Alfonso, A. Calatrava, and G. Molt, "Container-based Virtual Elastic Clusters," *Journal of Systems and Software*, vol. 127, no. C, pp. 1–11, May 2017.

[10] Docker Inc., "Docker: Enterprise Container Platform," https://www.docker.com/, last accessed in Dec 2018.

[11] The Cloud Native Computing Foundation, "Kubernetes: Production-Grade Container Orchestration," http://https://kubernetes.io/, last accessed in Dec 2018.

[12] The Jenkins Project, "Jenkins," https://jenkins-ci.org/, last accessed in Dec 2018.

[13] X. Zhou, X. Peng, T. Xie, J. Sun, W. Li, C. Ji, and D. Ding, "Delta Debugging Microservice Systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE '18)*. ACM, 2018, pp. 802–807.

[14] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The Journey So Far and Challenges Ahead," *IEEE Software*, vol. 35, no. 3, pp. 24–35, 2018.

[15] N. C. Mendonça, D. Garlan, B. Schmerl, and J. Cámara, "Generality vs. Reusability in Architecture-based Self-adaptation: The Case for Self-adaptive Microservices," in *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, ser. ECSA '18. ACM, 2018, pp. 18:1–18:6.

[16] H.-L. Truong, L. Gao, and M. Hammerer, "Service Architectures and Dynamic Solutions for Interoperability of IoT, Network Functions and Cloud Resources," in *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*. ACM, 2018, p. 2.

[17] A. Khan, "Key Characteristics of a Container Orchestration Platform to Enable a Modern Application," *IEEE Cloud Computing*, no. 5, pp. 42–48, 2017.

[18] S. Esparrachiari, T. Reilly, and A. Rentz, "Tracking and Controlling Microservice Dependencies," *Queue, ACM*, vol. 16, no. 4, p. 10, 2018. [Online]. Available: https://queue.acm.org/detail.cfm?id=3277541

[19] Buoyant Inc., "A Service Mesh for Kubernetes," last accessed in Dec, 2018. [Online]. Available: https://cdn2.hubspot.net/hubfs/2818724/A%20Service%20Mesh%20for%20Kubernetes_Final.pdf

[20] O. Sheikh, S. Dikaleh, D. Mistry, D. Pape, and C. Felix, "Modernize Digital Applications with Microservices Management using the Istio Service Mesh," in *Proceedings of the 28th Annual International Conference on Computer Science and Software Engineering (CASCON '18)*. IBM Corp., 2018, pp. 359–360.

[21] The Cloud Native Computing Foundation, "Linkerd: Production-grade Feature-rich Service Mesh for Any Platform." https://github.com/linkerd/linkerd, last accessed in Dec 2018.

[22] The Istio Team, "Istio: An Open Platform to Connect, Manage, and Secure Microservices." https://github.com/istio/istio, last accessed in Dec 2018.

[23] Amazon Web Services, Inc., "AWS App Mesh," https://aws.amazon.com/app-mesh/, last accessed in Dec 2018.

[24] Airbnb, "Synapse: A Transparent Service Discovery Framework for Connecting an SOA," https://github.com/airbnb/synapse, last accessed in Dec 2018.

[25] AliPay, "SOFAMesh: A Solution for Large-scale Service Mesh based on Istio," https://github.com/alipay/sofa-mesh, last accessed in Dec 2018.

[26] The Cloud Native Computing Foundation, "Envoy," https://www.envoyproxy.io/, last accessed in Dec 2018.

[27] AliPay, "MOSN Project," https://github.com/alipay/sofa-mosn, last accessed in Dec 2018.

[28] The Istio Team, "Mixer," https://github.com/istio/istio/tree/master/mixer, last accessed in Dec 2018.

[29] K. Indrasiri and P. Siriwardena, "Microservices Governance," in *Microservices for the Enterprise*. Springer, 2018, pp. 151–166.

[30] A. Edmonds, C. Woods, A. J. Ferrer, J. F. Ribera, and T. M. Bohnert, "Blip: JIT and Footloose on the Edge," *CoRR*, vol. abs/1806.00039, 2018.

[31] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan 2003.

[32] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center," in *Proceedings of 8th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI '11, vol. 11, 2011, pp. 22–22.

[33] J. Palm, "Service Isolation in Large Microservice Networks," June 11, 2018, Master Thesis in Computer Science, KTH Royal Institute of Technology, Sweden. [Online]. Available: http://www.nada.kth.se/~ann/exjobb/jonas_palm.pdf

[34] Peter Bourgon, "Metrics, Tracing, and Logging," last accessed in Dec, 2018. [Online]. Available: https://peter.bourgon.org/blog/2017/02/21/metrics-tracing-and-logging.html

[35] The Cloud Native Computing Foundation, "Prometheus: From Metrics to Insight," https://prometheus.io/, last accessed in Dec 2018.

[36] ——, "Jaeger: Open Source, End-to-End Distributed Tracing," http://jaegertracing.io, last accessed in Dec 2018.

[37] The Zipkin Project, " OpenZipkin: A Distributed Tracing System," https://zipkin.io/, last accessed in Dec 2018.

[38] The Cloud Native Computing Foundation, "Fluentd: Open Source Data Collector, Unified Logging Layer," https://www.fluentd.org/, last accessed in Dec 2018.

[39] ——, "Vendor-neutral APIs and Instrumentation for Distributed Tracing," https://opentracing.io/, last accessed in Dec 2018.

[40] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, "Performance Debugging for Distributed Systems of Black Boxes," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, ser. SOSP '03. ACM, 2003, pp. 74–89.

[41] P. Reynolds, J. L. Wiener, J. C. Mogul, M. K. Aguilera, and A. Vahdat, "WAP5: Black-box Performance Debugging for Wide-area Systems," in *Proceedings of the 15th International Conference on World Wide Web*, ser. WWW '06. ACM, 2006, pp. 347–356.

[42] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang, "Towards Highly Reliable Enterprise Network Services via Inference of Multi-level Dependencies," in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 13–24.

[43] M. Chow, D. Meisner, J. Flinn, D. Peek, and T. F. Wenisch, "The Mystery Machine: End-to-end Performance Analysis of Large-scale Internet Service," in *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*, ser. OSDI '14, 2014, pp. 217–231.

[44] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[45] M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen, "Orchestration of Microservices for IoT using Docker and Edge Computing," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 118–123, 2018.

[46] C. Pahl and B. Lee, "Containers and Clusters for Edge Cloud Architectures–A Technology Review," in *Proceedings of the 3rd International Conference on Future Internet of Things and Cloud*, ser. FiCloud '15. IEEE, 2015, pp. 379–386.

[47] K. Khanda, D. Salikhov, K. Gusmanov, M. Mazzara, and N. Mavridis, "Microservice-Based IoT for Smart Buildings," in *Proceedings of the 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, March 2017, pp. 302–308.

[48] B. Butzin, F. Golatowski, and D. Timmermann, "Microservices Approach for the Internet of Things," in *Proceedings of the IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2016, pp. 1–6.

[49] S. Shekhar and A. Gokhale, "Dynamic Resource Management across Cloud-edge Resources for Performance-sensitive Applications," in *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Press, 2017, pp. 707–710.

[50] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile Edge Computing - A Key Technology towards 5G," *ETSI White Paper*, vol. 11, no. 11, pp. 1–16, 2015. [Online]. Available: http://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp11_mec_a_key_technology_towards_5g.pdf

[51] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.