Sidecars on the Central Lane: Impact of Network Proxies on Microservices

Prateek Sahu
prateeks@utexas.edu
Univeristy of Texas at Austin

Lucy Zheng
lucy.zheng@utexas.edu
Univeristy of Texas at Austin

Marco Bueso
mbueso@utexas.edu
Univeristy of Texas at Austin

Shijia Wei
shijiawei@utexas.edu
Univeristy of Texas at Austin

Neeraja J. Yadwadkar
neeraja@austin.utexas.edu
Univeristy of Texas at Austin,
VMware Research

Mohit Tiwari
tiwari@austin.utexas.edu
Univeristy of Texas at Austin

ABSTRACT

Cloud applications are moving away from monolithic model towards loosely-coupled microservices designs. Service meshes are widely used for implementing microservices applications mainly because they provide a modular architecture for modern applications by separating operational features from application business logic. *Sidecar proxies* in service meshes enable this modularity by applying security, networking, and monitoring policies on the traffic to and from services. To implement these policies, sidecars often execute complex chains of logic that vary across associated applications and end up unevenly impacting the performance of the overall application. Lack of understanding of how the sidecars impact the performance of microservice-based applications stands in the way of building performant and resource-efficient applications. To this end, we bring sidecar proxies in focus and argue that we need to deeply study their impact on the system performance and resource utilization. We identify and describe challenges in characterizing sidecars, namely the need for microarchitectural metrics and comprehensive methodologies, and discuss research directions where such characterization will help in building efficient service mesh infrastructure for microservice applications.

KEYWORDS

Microservices, Sidecars, Service-Mesh, Performance

1 INTRODUCTION

Cloud applications are widely adopting loosely coupled microservice-based solutions [1–6]. With this change, operators of large cloud applications face increasing challenges in traffic management as these applications scale to hundreds, or even thousands of services [6–9]. The complex inter-service communication patterns of these microservices make it challenging to restrict, route, and monitor the generated traffic. Service meshes are widely adopted [10–13] as they provide robust frameworks for implementing and deploying tools that aid in simplifying the traffic management. Service meshes leverage *sidecar proxies* that can be configured to execute inter-service communication policies. These policies enable security, networking, and observability features such as endpoint access control, request rate limiting, and logging.

HotInfra'23, June 18, 2023, Orlando, FL, USA.
.

However, sidecars end up increasing request latency and result in performance penalties for large applications since they are designed to co-locate with each application container. Prior research [14] reports 30-185% increase in latency and 41-92% overhead in CPU usage for different benchmark applications. Other studies by leading service mesh vendors [15–17] note 2-6× latency overheads of the sidecar proxies. These studies also note that the CPU usage of sidecars ranges from negligible overhead to 0.35 vCPU for every 1000 requests based on the choice of sidecars and request rates.

Thus, it is crucial to understand the performance of sidecar proxies to be able to reason about the performance of microservice applications. However, characterizing the performance implications of sidecars with complex inter-service traffic management policies is challenging, since (a) no defined metrics exist, and (b) no established methodology exists to guide application operators and hardware architects to systematically optimize the performance of microservice applications. Existing efforts [2, 6, 14, 18, 19] are based on traditional metrics like latency and CPU or memory utilization to study microservice performance. However, these metrics provide no insights into how sidecar proxies interact with the underlying hardware. To the best of our knowledge, no prior profiling work has investigated the microarchitectural impacts of sidecar proxies in service mesh infrastructures. Additionally, we observe significant changes in application latencies depending on the types and complexity of policies used. However, studies from service mesh practitioners [15–17] neglect the impact of the diverse and complex set of network policies that are supported [20–22].

Our work brings sidecars into focus and argues for measuring their impact on application performance. In doing so, we highlight the lack of any microarchitectural metrics and omission of diverse network policies as two key challenges in understanding the performance overheads of service-mesh sidecars. We discuss how characterizing sidecar proxies—with microarchitectural metrics and a methodology covering diverse policies with varying complexity—enables application operators to navigate the performance trade-offs of service mesh infrastructures; and opens avenues in software and hardware research for microservices infrastructure.

2 BACKGROUND

To restrict, route, and monitor the inter-service traffic in microservice applications, service meshes deploy one sidecar proxy alongside each application process to mediate the corresponding traffic.

Specifically, sidecars apply policies to the network traffic, without modifying the main business logic or incurring any service downtime. Sidecars use a variety of listeners that implement a set of filters commonly known as filter-chains. Default filter-chains usually pass through packets with minimal logging and packet modifications. However, complex policies may require heavy computations, pattern matching, and request modification. For example, mTLS [23] enables mutually authenticating services by encrypting the inter-service traffic; application-layer role-based access control (RBAC) [24] enables fine-grained access control by restricting API access from certain services; request tagging [25] enables detailed telemetry collection by augmenting request headers along the service invocation chain. Furthermore, sidecar vendors provide operators with not only a long list of common filters [20–22] but also the option for writing custom filters [26, 27]. This configurability and programmability make the performance characteristics of a sidecar vary significantly across different deployment settings.

3 CHALLENGES

With the aforementioned complex policies, we discuss two major challenges in quantitatively navigating the performance space of employing sidecar proxies with diverse policies. Our experiments study the performance characteristic when an Envoy proxy [28] is allocated with different numbers of vCPUs, and when it is configured with different traffic policies. We explore two commonly used policies (a) RBAC (Role Based Access Control) and (b) IP Tag. The RBAC policy filters out incoming requests based on their source IP address, whereas the IP Tag policy incorporates specific header values to the HTTP request determined by source IP addresses.

Challenge 1: Lack of adequate metrics fails to highlight bottlenecks in sidecar proxies accurately.

Existing performance studies of service meshes [15–17] focus on user-sensitive metrics such as CPU utilization, latency, and throughput. However, Figure 1 shows without microarchitectural metrics insights like pipeline congestion using top-down analysis, operators cannot reason about performance improvement with allocation of additional OS resources. Figure 1 plots the latency and throughput behavior when an Envoy [28] proxy is allocated various vCPU time. We observe that allocation of two virtual cores (threads), mapped to the same physical core provides no performance increase over a single core allocated. We however do see throughput improve when we map it to separate cores. This behavior cannot be explained with system-level metrics and requires detailed insights into pipeline occupancy and logical unit contention to reason about the observed performance.

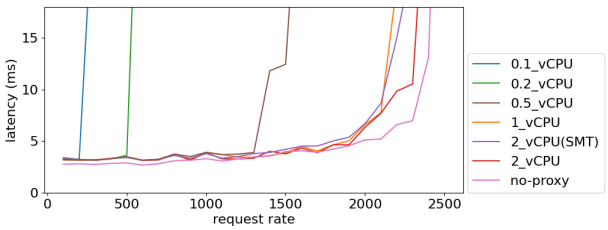

Figure 1: P90 latency for Envoy with increasing vCPU.

Policy	p90 Latency (ms)	p90 Cycles	Instructions
IP Tag (1)	1.034	1.035	1.052
IP Tag (5)	1.039	1.054	1.088
IP Tag (10)	1.048	1.108	1.147
RBAC (100)	1.029	1.123	1.014
RBAC (10k)	1.044	1.137	1.014

Table 1: Latency, cycle, and instruction overhead for two policies (normalized to baseline with no policies) - HTTP request tagging with 1, 5, 10 tags , Role-Based Access Control (RBAC) with 100 and 10k rules respectively.

Similarly, latency and dynamic instruction count of the IP Tag policy in Table 1 show another example of the need of microarchitectural metrics for attributing the hardware performance bottleneck. We notice that increasing the number of tags in IP Tagging from 1 to 5, and 10 results in a linear increase in dynamic instructions. However, over the 1-tag baseline, cycle counts only increase by 1.8% for 5 tags yet by 7% for 10 tags. A deeper investigation reveals that this non-linear overhead is due to L2-cache misses incurred when processing 10 tags. IP Tagging with 10 tags shows a nearly 10% overhead in L2 misses while processing 5 tags incurs 2.1%.

Challenge 2: Neglecting network policies during profiling misleads performance trend prediction for sidecar proxies.

Most of service-mesh performance studies [29, 30] follow practices in microservices benchmarking [6, 18, 19], which focuses on the impact of request sizes and request rates on performance metrics over a representative application set. Although they remain relevant aspects to study the performance impact of sidecars, the methodology needs to include diverse policies with varying complexity to provide a comprehensive analysis. Prior work by Zhu *et al.* [14] characterizes the overhead for some filters but omits policy complexities and their interactions with the microarchitecture.

Table 1 shows that different policies have distinct performance impact on application latency, and increasing the policy complexity affects performance differently. In Table 1, we note that application latency for both policies are similar but the instruction footprint is significantly higher in IP tagging. Differences in execution profiles and the lack of a representative set of filters is a major challenge to predict performance costs.

4 RESEARCH DIRECTIONS

By design, sidecars are central places to implement and consolidate common operational tasks, also known as the 'datacenter tax'. Kanev *et al.* [31] suggested that this datacenter tax, such as protocol management, remote procedure calls, and data movement contributes over 20% of CPU cycles. Their characterization of microarchitectural fetch latency and cache misses motivates further research in systematically understanding the performance of the operational tasks. As tasks around networking, telemetry, and security get consolidated, sidecars offer unique opportunities for software and hardware innovations. In this section, we outline how microarchitectural metrics and a methodology that covers diverse policies with varying complexity could benefit software and hardware research in service-mesh infrastructures.

Performance prediction and optimization in service mesh. Our characterization enables application operators to reason about

performance trends of diverse policies. Inclusion of microarchitectural metrics in profilers for sidecar proxies allows us to build automated and dynamic tools to predict and optimize service-mesh infrastructure for improved performance and hardware utilization. Such tools would enable predictable service-mesh performance and improved system utilization while maintaining the desired quality of service.

Hardware support for service-mesh infrastructures. Hardware vendors are designing solutions [32, 33] to accelerate several cloud infrastructure components including network and storage. Our characterization of the microarchitectural implications of sidecar proxies with a comprehensive suite of network policies helps architects design specialized hardware that further offload the sidecar infrastructure efficiently. Offloading such service-mesh components reduces interference, and thus enables scalable microservice applications.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their valuable and constructive feedback. This work was supported in part by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA and by Intel RARE grant.

REFERENCES

[1] S. Joyner, M. MacCoss, C. Delimitrou, and H. Weatherspoon, "Ripple: A Practical Declarative Programming Framework for Serverless Compute," in *arXiv:2001.00222 [cs.DC]*, January 2020.

[2] Y. Zhang, W. Hua, Z. Zhou, E. Suh, and C. Delimitrou, "Sinan: Data-Driven Resource Management for Interactive Microservices," in *Workshop on ML for Computer Architecture and Systems (MLArchSys)*, June 2020.

[3] Y. Gan, S. Dev, D. Lo, and C. Delimitrou, "Sage: Leveraging ML To Diagnose Unpredictable Performance in Cloud Microservices," in *Workshop on ML for Computer Architecture and Systems (MLArchSys)*, June 2020.

[4] M. Shahrad, J. Balkind, and D. Wentzlaff, "Architectural implications of function-as-a-service computing," in *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*, 2019, pp. 1063–1075.

[5] Y. Gan and C. Delimitrou, "The Architectural Implications of Cloud Microservices," in *Computer Architecture Letters (CAL), vol.17, iss. 2*, Jul-Dec 2018.

[6] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson *et al.*, "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 3–18.

[7] "Netflix architecture: How much does netflix's aws cost?" https://www.cloudzero.com/blog/netflix-aws.

[8] "Lyft runs 300,000+ containers in a multicluster kubernetes environment | altoros," https://www.altoros.com/blog/lyft-runs-300000-containers-in-a-multicluster-kubernetes-environment/.

[9] Z. Zhang, M. K. Ramanathan, P. Raj, A. Parwal, T. Sherwood, and M. Chabbi, "{CRISP}: Critical path analysis of {Large-Scale} microservice architectures," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022, pp. 655–672.

[10] "Cncf_survey_report_2020.pdf," https://www.cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf.

[11] "Istio / case studies," https://istio.io/latest/about/case-studies/.

[12] "Cilium users and real world case studies," https://cilium.io/adopters/.

[13] "Linkerd 2.x adopters | linkerd," https://linkerd.io/community/adopters/.

[14] X. Zhu, G. She, B. Xue, Y. Zhang, Y. Zhang, X. K. Zou, X. Duan, P. He, A. Krishnamurthy, M. Lentz *et al.*, "Dissecting service mesh overheads," *arXiv preprint arXiv:2207.00592*, 2022.

[15] "Istioldie 1.11 / performance and scalability," https://istio.io/v1.11/docs/ops/deployment/performance-and-scalability/.

[16] "Benchmarking linkerd and istio: 2021 redux | linkerd," https://linkerd.io/2021/11/29/linkerd-vs-istio-benchmarks-2021/.

[17] "Cni benchmark: Understanding cilium network performance," https://cilium.io/blog/2021/05/11/cni-benchmark/, (Accessed on 05/02/2023).

[18] A. Sriraman and T. F. Wenisch, "μ suite: a benchmark suite for microservices," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2018, pp. 1–12.

[19] S. Chen, C. Delimitrou, and J. F. Martínez, "Parties: Qos-aware resource partitioning for multiple interactive services," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 107–120. [Online]. Available: https://doi.org/10.1145/3297858.3304005

[20] "Network filters — envoy 1.27.0-dev-f2a6dc documentation," https://www.envoyproxy.io/docs/envoy/latest/configuration/listeners/network_filters/network_filters.

[21] "Http filters — envoy 1.27.0-dev-f2a6dc documentation," https://www.envoyproxy.io/docs/envoy/latest/configuration/http/http_filters/http_filters.

[22] "Network policy — cilium 1.13.2 documentation," https://docs.cilium.io/en/stable/security/policy/.

[23] "Rfc 8705 - oauth 2.0 mutual-tls client authentication and certificate-bound access tokens," https://datatracker.ietf.org/doc/html/rfc8705.

[24] "Role based access control (rbac) filter — envoy 1.27.0-dev-70be00 documentation," https://www.envoyproxy.io/docs/envoy/latest/configuration/http/http_filters/rbac_filter.

[25] "Ip tagging — envoy 1.27.0-dev-70be00 documentation," https://www.envoyproxy.io/docs/envoy/latest/configuration/http/http_filters/ip_tagging_filter.

[26] "Wasm — envoy 1.27.0-dev-f2a6dc documentation," https://www.envoyproxy.io/docs/envoy/latest/configuration/http/http_filters/wasm_filter.

[27] "Lua — envoy 1.27.0-dev-f2a6dc documentation," https://www.envoyproxy.io/docs/envoy/latest/configuration/http/http_filters/lua_filter.

[28] "Envoy proxy - home," https://www.envoyproxy.io/.

[29] "Standardizing service mesh value measurement," https://smp-spec.io/.

[30] "Service mesh performance evaluation — istio, linkerd, kuma and consul | by florent martin (elca) | elca it | medium," https://medium.com/elca-it/service-mesh-performance-evaluation-istio-linkerd-kuma-and-consul-d8a89390d630.

[31] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a warehouse-scale computer," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ser. ISCA '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 158–169. [Online]. Available: https://doi.org/10.1145/2749469.2750392

[32] B. Burres, D. Daly, M. Debbage, E. Louzoun, C. Severns-Williams, N. Sundar, N. Turbovich, B. Wolford, and Y. Li, "Intel's hyperscale-ready infrastructure processing unit (ipu)," in *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 2021, pp. 1–16.

[33] I. Burstein, "Nvidia data center processing unit (dpu) architecture," in *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 2021, pp. 1–20.