Stochastic Modelling and Computational Sciences

OPTIMIZING MICROSERVICES FOR SCALABLE ENTERPRISE APPLICATIONS IN CLOUD COMPUTING

Karthik Sirigiri[1*] and Akhila Narra[2]
[1]Software Developer, Euniverse Technologies, Irving, Texas, USA
[2]Full Stack Developer, AT&T, Irving, Texas, USA
[1]sirigirikarthik25@gmail.com and [2]narraakhila242@gmail.com

ABSTRACT
Microservices' design has fundamentally transformed the approach scalable and robust commercial applications are produced in the world of cloud computing. Still, the search for ideal scalability challenges service orchestration, data consistency, network latency, and resource management. This work offers a theoretical analysis of optimization techniques meant to enhance the microservices' performance in cloud systems. Examined mostly are service design patterns, container orchestration with Kubernetes, management of API gateways, and database optimization. The study emphasizes the need of thorough observability procedures since it investigates the complex difficulties in monitoring and debugging distributed microservices. Furthermore included in the paper are emerging topics such edge computing's ideas and artificial intelligence-driven performance improvement. This paper aims to integrate industry best practices with current research so that a comprehensive knowledge of how businesses should maximize microservices architecture is given. Achieving excellent performance, economy, and dependability inside cloud-native systems takes front stage.

Keywords: *Microservices Architecture, API Gateway, Serverless Computing, Enterprise Applications, Cloud Computing, Distributed Systems, Kubernetes, Service Optimization, Performance Monitoring, Database sharding.*

INTRODUCTION
When compared with traditional monolithic systems, microservices' architecture has greatly changed the terrain of corporate application development and deployment by offering benefits in modularity, scalability, and resilience. By breaking out applications into independently deployable services, companies can achieve improved agility and a faster time to market. The growing popularity of cloud computing has made microservices a basic part of modern systems, helping companies to reach both high availability and flexibility.

Still, the attempt to maximize microservices for scalability presents significant difficulties covering areas including service orchestration, network latency, database consistency, and resource allocation. While the management of distributed data consistency calls for the application of advanced techniques, the overhead related with inter-service communication has the potential to produce major performance bottlenecks. Moreover, the guarantee of effective use of resources in a dynamic cloud environment calls for the application of strong monitoring and orchestration instruments.

This work explores theoretical optimization techniques meant to increase microservices architecture scalability and efficiency in cloud computing systems. Discourse mostly covers service design patterns, container orchestration using Kubernetes, management of API gateways, database optimization, and performance monitoring. This study clarifies outstanding industry practices, offers case studies from well-known companies, and investigates developing trends including the optimization of microservices driven by artificial intelligence and the developments in edge computing.

This paper aims to consolidate current research and business strategies so providing a complete knowledge of how companies could maximize microservices architecture to achieve high performance, cost-effectiveness, and dependability inside cloud-native environments.

ARCHITECTURAL FUNDAMENTALS OF MICROSERVICES

In software development, microservices architecture is the method of splitting the big systems into more small, manageable, specialized components then arranged utilizing a network architecture. This method displays a sophisticated variation from the typical monolithic systems, in which the linked character of the components causes limits of adaptability and Scalability.

KEY PRINCIPLES AND CHARACTERISTICS

Each and every microservice is deliberately designed to serve a specific, well-defined purpose in line with the Single Responsibility concept, therefore enhancing modularity and supporting maintenance. This model allows every service to run its own independent database rather than depending on a central database. The relevant system can grow more independent and scalable with this method's assistance.

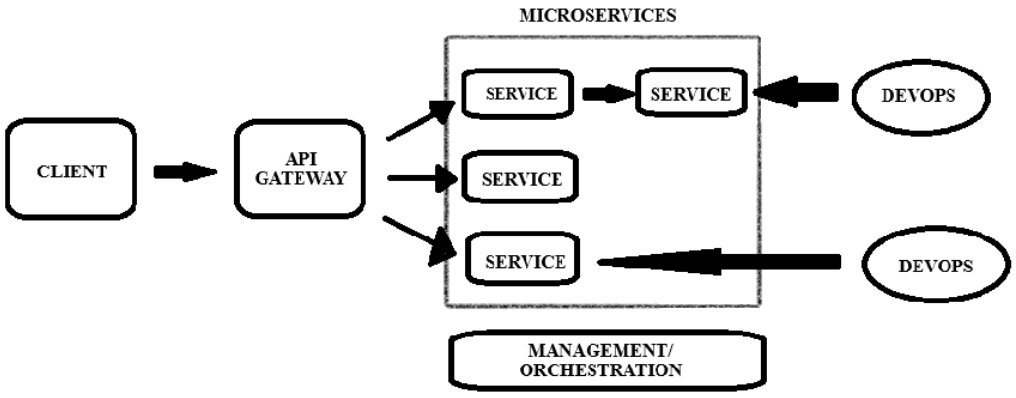
By means of lightweight communication technologies like REST or gRPC, services interact, therefore promoting adaptation and interoperability. With a specific focus on Continuous Integration and Continuous Deployment (CI/CD) pipelines, the efficient administration of microservices is essentially dependent on the execution of Continuous Deployment and Automation through the application of DevOps methodologies.

ANALYTICAL COMPARISON OF MONOLITHIC BUILDING STYLES

Although traditional monolithic systems limit scalability, they improve development by combining all functionality into a single codebase. Microservices have many natural benefits:

- Scalability is the capacity of specific services to be scaled independently, hence improving the effectiveness of resource use.
- By guaranteeing that a failure inside one microservice does not inevitably compromise the general functionality of the whole program, fault isolation helps to increase the resilience of a system.
- Thanks to its adaptability, several programming languages, frameworks, and database, each especially tailored to fit the particular requirements of different service that can be used. This flexibility helps teams choose the best tools for every given job.

Despite its benefits, microservices include increased network overhead and the need for Kubernetes as a service orchestration tool which bring great complexity.



BENEFITS & DIFFICULTIES

Benefits of Microservices:

Teams enabled by microservices' agility and acceleration of development processes have the capacity to independently create, test, and implement services. Eventually, this autonomy results in shorter time-to-market and less development cycles.

The improvement of resource allocation and cost effectiveness defines a characteristic of cloud-native architectures, which enable the scalability of services in line with demand.

Difficulties of Microservices:
The management of several autonomous services calls for the development of advanced orchestration techniques, therefore adding complexity to the monitoring and deployment processes.

Maintaining consistency in data integrity calls across databases will be difficult. This leads for the adoption of sophisticated techniques such event-driven architecture.

TECHNIQUES OF SCALABILITY IN MICROSERVICES ARCHITECTURE

Horizontal and vertical scaling

Microservices-based systems show both vertical scalability—that is, the augmentation of resources such as CPU and memory associated to a single instance as well as horizontal scaling, that is the rise in the number of service instances. Because horizontal scaling allows workloads to be effectively distributed over several nodes, it is desirable. Still, vertical scaling could show advantages in reaching quick performance gains without having to add extra cases. Studies indicate that until a certain point—above which horizontal scaling shows more efficiency—vertical scaling is still cost-effective.

Load Distribution.

Load balancing systems help to distribute demands among microservices, therefore lowering the possibility of system overload. Round-robin, least connections, and randomized load balancing among other algorithms greatly increase request distribution's effectiveness. Modern cloud-native systems best represented by Kubernetes allow load balancing to be automated using technologies including service discovery and ingress controllers. Studies show that inadequate load balancing techniques can seriously affect performance, particularly in view of irregular traffic surges.

Techniques of Caching

By storing often accessed data within memory resources, caching features help to lower response times and increase scalability. Among the several techniques that may be used, one could consider in-memory caching best shown by Redis in addition to distributed caching and HTTP caching especially intended for API answers. Analyzing MuCache, a microservices caching system, shows that using caching resulted in a notable drop in median request latency up to 2.5 times, while concurrently enabling an amazing gain in throughput by 60%. Still, inadequate cache invalidation can result in stale data problems.

Asynchronous Computing and Event-Driven Design

By separating services, event-driven designs greatly increase scalability. Using message queues like Apache Kafka and RabbitMQ in concert with event sources helps to achieve this decoupling. This helps services to do chores on their own, therefore reducing any system congestion. Research shows that event-driven microservices greatly increase distributed architectures' resilience and efficiency. Still, the management of event sequencing and the attainment of ultimate consistency could provide somewhat difficult problems.

Approaches to Reach Database Scalability

Every microservice follows the common habit of keeping its own database, which helps to lower dependencies and advance scalability. To improve performance during periods of higher demand, methods include sharding—that is, the division of databases into smaller segments and replication—that is the production of read replicas are used. Research indicates that the application of polyglot persistence, defined by the use of several databases for particular services, greatly increases system flexibility.

Service Mesh's Use in Enabling Inter-service Communication

By abstracting the complexities of inter-service communication, a service mesh guarantees that the traffic among microservices is safe, efficient, and observable. Technologies that improve dependability include dynamic

routing, automated retries, and circuit breaking seen in Istio and Linkerd. Essential for the scaling of challenging microservice architectures, service mesh frameworks help to lower communication overhead while enhancing system observability.

Policies for Auto-Scaling and Natural Solutions Based on the clouds

Designed to dynamically change resource allocation in response to CPU use, memory use, or changing rates of incoming requests, auto-scaling policies are. Together with cloud-based auto-scaling systems, the horizontal pod autoscaler (HPA) included in the Kubernetes ecosystem offers a noteworthy degree of elasticity for applications developed with a microservices architecture. By anticipating workload patterns, auto-scalers using machine learning approaches have proven the capacity to maximize resource allocation, hence improving cost efficiency.

MICROSERVICES IN CLOUD COMPUTING: METHODS OF OPTIMIZATION

Orchestration's and containerizing's roles

Within the microservices space, containerizing helps to create isolated and lightweight environments, hence improving scalability and mobility. By use of Docker and Kubernetes, developers may achieve automated deployment, guarantee resource isolation, and enable effective scaling. Orchestration solutions like Kubernetes really help to automate important tasks as load balancing, failover management, and resource scheduling. Studies show that using containerized microservices improves fault tolerance, which also helps to lower response times. Kubernetes scheduling's application of queueing models helps to improve resource allocation in line with real-time needs. This approach reduces resource consumption and greatly improves operational effectiveness.

Function-as- a- Service (FaaS)

A cloud computing architecture called Function-as- a- Service (FaaS) lets developers implement specific business logic or functions free from the complications of running the underlying infrastructure. By allowing a more modular approach to application development, this paradigm helps to enable scalability and resource efficiency. Mini services by means of Function-as- a- Service (FaaS), microservices can be deployed on an on-demand basis, therefore enabling independent scaling without human involvement. The operational load is considerably reduced by the abstraction of server management made possible by AWS Lambda and Azure Functions. Operating just when needed, serverless microservices enable a sophisticated way to scaling, hence reducing costs. Still, it's essential to solve latency problems and cold starts to guarantee best performance in high-performance systems.

Database Optimization to Boost Microservices Architectural Scalability

Following the Database-per- Service design, each microservice usually manages its own database, hence improving modularity and helping fault isolation. Designed to meet particular service needs, the idea of polyglot persistence helps to use several database systems, including both SQL and NoSQL, so improving general efficiency. Scaling database workloads and concurrently lowering query latency depend much on sharding and replication. In the framework of distributed transactions, the Saga pattern provides a means to preserve data consistency.

Service Mesh's Function in Promoting Inter-Service Communication

While preserving microservices' integrity, service meshes including Istio, Linkerd, and Consul Connect improve network efficiency, security, and observability without calling for changes. By means of traffic flow and request routing, which helps to reduce both latency and the frequency of errors, network overhead is minimized. Service meshes enable canary releases and traffic shaping, therefore allowing safe and slow deployments supported by real-time monitoring. While fine-grained access limits illegal interactions between services, mutual TLS (mTLS) encryption helps to improve security by facilitating safe communication. Prometheus, Grafana, and OpenTelemetry jointly provide real-time metrics and traces necessary for the optimization of performance, hence enhancing observability. Service meshes reduce complexity and improve microservices' performance by means of traffic management, security mechanisms, and monitoring systems optimization.

Intelligent Scaling Methodologies and Auto-Scaling

For scaling, conventional auto-scalers such as the Kubernetes Horizontal Pod Autoscaler (HPA) rely on CPU and memory measurements. Modern approaches, however, use machine learning tools to project growth needs and improve service availability. By considering the interactions across several services, holistic auto-scaling reduces the possibility of bottlenecks resulting from increasing individual services alone. Studies show that using intelligent predictive scaling not only reduces costs but also guarantees the preservation of ideal performance levels.

Observability and Performance Monitoring

Microservices depend on constant monitoring to find performance criteria violations and bottlenecks. Prometheus and Open Telemetry provide real-time statistics; Grafana helps to show the state of services. Tools like Jaeger and Zipkin help to monitor requests moving across microservices, therefore supporting both debugging procedures and performance optimization by distributed tracing. According to a recent analysis, using automated monitoring inside Kubernetes has improved response times and resource efficiency equally.

Microservices Deployment Strategies Analysis at Uber and Netflix

From monolithic designs to microservices, Netflix and Uber have deftly changed to produce increased scalability, resilience, and operational efficiency in cloud-native systems. Their encounters help to clarify the advantages and challenges of microservices inside large systems.

Early on, Netflix was a single program that had major difficulties with scalability and stability as its global user base expanded. A large system breakdown in 2008 exposed the inherent flaws in this design, which prompted a move toward a microservices-based architecture using Amazon Web Services (AWS. The change helped Netflix to be able to independently increase the range of services it provides and use horizontal scalability, therefore guaranteeing consistent content delivery throughout different geographical areas. With around one billion microservice calls daily in 2016, Netflix demonstrated the amazing effectiveness of their just adopted architectural framework.

Microservices have enabled Netflix to improve resilience by means of fault isolation. Individual services can fail without upsetting the whole system. Using Hystrix, Netflix has used the circuit breaker pattern to reduce the risks connected with cascade failures thereby assuring that any degradation in service does not impact the general performance of the system. Moreover, Netflix leads in chaotic engineering using tools like chaotic Monkey to methodically evaluate their system robustness. The above mentioned developments have greatly enhanced an architectural framework able to accommodate over 200 million users, skillfully control unanticipated traffic spikes while preserving high availability.

The microservices approach Netflix has embraced has improved developers' agility. Engineers show an amazing ability to repeatedly apply improvements many times during the day while yet preserving system-wide operational continuity. For new features, this approach greatly speeds up the time-to--market. As shown by the adaptive changes done to the recommendation engine at times of maximum usage, this modular design helped Netflix to be able to expand and improve its services in response to demand.

Uber started its business with a monolithic design; yet, when its ride-sharing network expanded to include more than 400 locations, this structure became a major obstacle. The system had major scaling difficulties that hampered its capacity to manage the increasing ride demand and enable global growth. Uber has adopted a microservices architecture, methodically disassembling basic services including ride-matching, payments, and driver management into autonomous entities. Every service can be scaled individually, therefore enabling elastic resource allocation sensitive to real-time demand.

Using Apache Kafka, Uber has developed an event-driven architecture that guarantees ultimate consistency across distributed databases and helps asynchronous communication among services. This architectural method reduces

system-wide failures resulting from dependencies inherent in the monolithic construction so improving dependability.

With each service supervised by specialized, specialized engineering teams, Uber's architectural framework as of 2022 had grown to include over 2,200 microservices.

Microservices adoption at Uber offers several benefits; but, they have also resulted in operational complexity increase. Debugging and controlling dependencies across several services has given somewhat difficult problems. Uber has chosen a Domain-Oriented Microservices Architecture (DOMA) to systematically arrange linked services, so improving system maintainability and reducing inter-service latency. Uber was able to guarantee the dependability of its services and achieve effective scalability using this methodical approach at the same time.

Microservices' transforming power in running large-scale operational systems is best shown by Netflix and Uber. While Uber has used microservices to provide real-time ride-matching in response to growing demand, Netflix has used them to improve its capability for resilient and worldwide content distribution. Their work clarifies basic ideas in fault isolation, scalability, event-driven communication, and operational complexity management.

Future Trends and Research Directions

Using artificial intelligence and machine learning is greatly enhancing the capacity of auto-scaling in microservices systems. While artificial intelligence may predict trends and assign resources in a proactive way, conventional scaling techniques react to load changes. When compared with the default auto-scaler of Kubernetes, reinforcement learning improves the optimization of scaling operations and results in a notable 20% reduction in latency. Predictive models are meant to lower operating costs at the same time by mitigating the hazards connected with over-provisioning.

By means of autonomous scalability, serverless computing offers the benefit of removing the requirement for human involvement. Function as a Service (FaaS) works by running services just when needed, hence improving scalability in reaction to changing demand. Dynamic resource allocation via AWS Lambda and Azure Functions corresponds with different demand levels. Comparative studies show that whilst containerized solutions are more suitable for jobs requiring continuous and steady processing, Function as a Service (FaaS) delivers better benefits for workloads marked by intermittent activity. Cold-start latency still presents a major obstacle, but, improvements in provisioning are clearly reducing execution delays.

The communication dynamics among different services are much enhanced by the service mesh. Managing routing, security, and telemetry depends mostly on Istio and Linkerd, hence improving the resilience of distributed systems. It is important to understand that sidecar proxies might cause latency, which might raise response times of up to 269%. To address this issue, research on proxy-less service meshes applying eBPF looks into eBPF.

Reducing microservices latency by means of strategic data processing close to customers depends on edge computing in great part. For real-time applications—including analytics and autonomous systems—this proves helpful. Workload management in cloud and edge systems offers several difficulties, which calls for careful investigation of dynamic migration solutions. Currently under research is the use of reinforcement learning to enhance microservices placement.

The security problem is really difficult and mostly results from the enlarged attack surfaces that have surfaced. The enforcement of security rules depends much on API gateways and service meshes; nonetheless, there are still rather clear issues with identity management. Using AI-driven security rules together with anomaly detection technologies greatly increases microservices' integrity. Observability is quite important since the ability for real-time monitoring helps to spot hazards before they become more serious.

One major obstacle of stateful microservices' scalability is Cross-cloud systems provide complexity to load balancing and service discovery. Artificial intelligence driven orchestration helps microservices to optimize themselves, hence improving scalability and performance.

Modern research focuses on reaching a balance among security, performance, and scalability.

CONCLUSION

The microservices architecture in cloud computing made the applications resilient, agile and scalable. Their modular design substantially speeds up development cycles and allows autonomous scaling; so, the significance of this architecture in contemporary cloud-native apps is very great. Microservices' operational performance has to be improved by means of serverless computing, service mesh implementation, and artificial intelligence-informed auto-scaling mechanism adoption. Realizing the importance of edge computing for applications requiring real-time processing can help one to get even more low latency. Still, challenges exist in fields such security, observability, and stateful system management. Solving these problems and fully using microservices in big-scale corporate systems depend on creative research and adaptable design techniques. Microservices-based cloud computing systems must show longevity as well as scalability. Recent advances in intelligent automation, cross-cloud installations, and improved microservices orchestration point to bright future directions in these fields.

REFERENCES

[1] Ganesh Chowdary Desina. Evaluating the impact of cloud-based microservices architecture on application performance. arXiv preprint arXiv:2305.15438, 2023.

[2] Rong Zeng, Xiaofeng Hou, Lu Zhang, Chao Li, Wenli Zheng, and Minyi Guo. Performance optimization for cloud computing systems in the microservice era: state-of-the-art and research opportunities. Frontiers of Computer Science, 16(6):166106, 2022.

[3] Guozhi Liu, Bi Huang, Zhihong Liang, Minmin Qin, Hua Zhou, and Zhang Li. Microservices: architecture, container, and challenges. In 2020 IEEE 20th international conference on software quality, reliability and security companion (QRS-C), pages 629–635. IEEE, 2020.

[4] Mehmet S¨oylemez, Bedir Tekinerdogan, and Ay¸ca Kolukısa Tarhan. Challenges and solution directions of microservice architectures: A systematic literature review. Applied sciences, 12(11):5507, 2022.

[5] Sasa Baskarada, Vivian Nguyen, and Andy Koronios. Architecting microservices: Practical opportunities and challenges. Journal of Computer Information Systems, 60:1–9, 09 2018.

[6] Grzegorz Blinowski, Anna Ojdowska, and Adam Przybylek. Monolithic vs. microservice architecture: A performance and scalability evaluation. IEEE Access, 10:20357–20374, 2022.

[7] Paolo Francesco, Patricia Lago, and Ivano Malavolta. Architecting with microservices: a systematic mapping study. Journal of Systems and Software, 150, 04 2019.

[8] Priyanka Patel, Priyesh Shah, and N. Ramani. Microservices: Architecture and technologies. International Journal for Research in Applied Science & Engineering Technology (IJRASET), 8:882–886, 2020.

[9] Israel Quintanilla, Joao Ejarque, and Rosa M. Badia. Improving microservice based applications with runtime placement adaptation. Journal of Internet Services and Applications, 10(1):1–15, 2019.

[10] Omar Al-Debagy and Miroslav Martinek. On microservice analysis and architecture evolution: A systematic mapping study. Applied Sciences, 11(17):7856, 2021.

[11] Hao Wang, Yong Wang, Guanying Liang, Yunfan Gao, Weijian Gao, and Wenping Zhang. Research on load balancing technology for microservice architecture. In MATEC Web of Conferences, volume 336, page 08002, 2021.

[12] Chao Qu, Rodrigo N. Calheiros, and Rajkumar Buyya. Auto-scaling web applications in clouds: A taxonomy and survey. ACM Computing Surveys, 51(4):73:1–73:33, 2018.

[13] Ali Goli, Nima Mahmoudi, Hamzeh Khazaei, and Omid Ardakanian. A holistic machine learning-based autoscaling approach for microservice applications. In Proc. of the 11th Int. Conf. on Cloud Computing and Services Science (CLOSER), pages 190–198, 2021.

[14] Bayya, Anil Kumar. (2022). Cutting-Edge Practices for Securing APIs in FinTech: Implementing Adaptive Security Models and Zero Trust Architecture. International journal of applied engineering and technology (London). 4. 279-298.

[15] Rabin Raj Gautam, Daya Sagar Baral, and Ganesh Gautam. Optimizing microservice communications: A hybrid service mesh approach merging linkerd and istio with maglev hashing. In Proc. of the 14th IOE Graduate Conference, pages 394–401, 2023.

[16] Oren Eini. Microservices and nosql: A great match. The New Stack, 2019.

[17] Pranita Tupsakhare. Monolithic vs. microservices: Analyzing architectural paradigms in modern software design. Journal of Scientific and Engineering Research, 6(2):299–303, 2019.

[18] Vamsi Krishna Yepuri, Venkata Kalyan Polamarasetty, Shivani Donthi, and Ajay Kumar Reddy Gondi. Containerization of a polyglot microservice application using docker and kubernetes. arXiv preprint arXiv:2305.00600, 2023.

[19] Erwin van Eyk, Alexandru Iosup, Johannes Grohmann, Simon Eismann, Andr´e Bauer, Laurens Versluis, Lucian Toader, Norbert Schmitt, Nikolas Herbst, and Cristina L. Abad. The SPEC-RG Reference Architecture for FaaS: From Microservices and Containers to Serverless Platforms. IEEE Internet Computing, 23(6):7–18, 2019.

[20] Mohamed Hedi Fourati, Soumaya Marzouk, and Mohamed Jmaiel. Towards microservices-aware autoscaling: A review. In 2023 IEEE Symposium on Computers and Communications (ISCC), pages 1080–1083. IEEE, 2023.

[21] Muhammad Usman, Simone Ferlin, Anna Brunstrom, and Javid Taheri. A survey on observability of distributed edge & container-based microservices. IEEE Access, 10:86905–86932, 2022.

[22] Freddy Tapia, Miguel 'A. Mora, Walter Fuertes, Hern'an Aules, Edwin Flores, and Theofilos Toulkeridis. From monolithic systems to microservices: A comparative study of performance. Applied Sciences, 10(17):5797, 2020.

[23] Tom Killalea. The hidden dividends of microservices. Communications of the ACM, 59(8):42–45, 2016.

[24] K. S. Swarnalatha, Adithya Mallya, G. Mukund, and R. Ujwal Bharadwaj. Solving problems of large codebases: Uber's approach using microservice architecture. In Emerging Research in Computing, Information, Communication and Applications, volume 928 of Lecture Notes in Electrical Engineering, pages 653–662. Springer, Singapore, 2022.

[25] Silvia Esparrachiari, Tanya Reilly, and Ashleigh Rentz. Tracking and controlling microservice dependencies. Communications of the ACM, 61(11):98–104, 2018.

[26] Adam Rubak and Javid Taheri. Machine learning for predictive resource scaling of microservices on kubernetes platforms. In Proceedings of the 16th IEEE/ACM International Conference on Utility and Cloud Computing (UCC '23), 2023.

[27] Abeer Abdel Khaleq and Ilkyeun Ra. Intelligent autoscaling of microservices in the cloud for real-time applications. IEEE Access, 9:35464–35476, 2021.

[28] Chen-Fu Fan, Anshul Jindal, and Michael Gerndt. Microservices vs serverless: A performance comparison on a cloud-native web application. In Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER 2020), pages 204–215, 2020.

[29] Kim Long Ngo, Joydeep Mukherjee, Zhen Ming Jiang, and Marin Litoiu. Evaluating the scalability and elasticity of function as a service platform. In Proceedings of the ACM/SPEC International Conference on Performance Engineering (ICPE ’22), 2022.

[30] Xiangfeng Zhu, Guozhen She, Bowen Xue, Yu Zhang, Yongsu Zhang, Xuan Kelvin Zou, Xiongchun Duan, Peng He, Arvind Krishnamurthy, Matthew Lentz, Danyang Zhuo, and Ratul Mahajan. Dissecting overheads of service mesh sidecars. In Proceedings of the ACM Symposium on Cloud Computing (SoCC ’23), 2023.

[31] Kaustabha Ray, Ansuman Banerjee, and Nanjangud C. Narendra. Proactive microservice placement and migration for mobile edge computing. In Proceedings of the IEEE/ACM Symposium on Edge Computing (SEC 2020), 2020.

[32] Davide Berardi, Saverio Giallorenzo, Jacopo Mauro, Andrea Melis, Fabrizio Montesi, and Marco Prandini. Microservice security: a systematic literature review. PeerJ Computer Science, 8:e779, 2022.