International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Serverless Cloud Computing: Advantages, Disadvantages

***Parul Tyagi*[1], *Vanshika Vats*[2], *Vikram Kumbhakar*[3], *Vishal Kumar*[4], *Vicky Kumar Matho*[5]**

[1]Assistant Professor, Computer Science and Engineering, Quantum University, Roorkee, India;
[2-5] B. Tech Student, Computer Science and Engineering, Quantum University, Roorkee, India;

ABSTRACT

Serverless computing has emerged as a transformative paradigm within the realm of cloud technology, redefining how applications are designed, deployed, and maintained. By abstracting away infrastructure management, it allows developers to focus exclusively on writing and improving application logic, while cloud providers dynamically allocate the necessary resources. This model primarily realized through Function as a Service (FaaS) and Backend as a Service (BaaS) offers significant advantages such as cost efficiency, automatic scalability, faster development cycles, and minimal operational overhead. Moreover, its pay-per-use billing model ensures that organizations only incur costs based on actual resource consumption, enhancing both agility and economic sustainability. Despite these benefits, several challenges persist, including cold start latency, potential vendor lock-in, limited configuration flexibility, and emerging security concerns. With applications spanning data processing, web development, IoT backends, and real-time analytics, serverless computing is rapidly becoming a cornerstone of modern cloud ecosystems. As the technology continues to evolve, further research and optimization are vital to address its current limitations and to fully realize its potential as the foundation of next-generation cloud computing.

Introduction

Serverless computing is a cloud deployment model in which the cloud provider dynamically manages server resources, allowing developers to deploy and run code (functions) without explicit provisioning of machines. In this paradigm, the cloud user writes and submits functions in response to events, and the provider automatically allocates resources to execute those functions, charging only for the exact execution time and memory used. In effect, serverless "handles virtually all the system administration operations" for the developer. Despite the name, "serverless" still relies on servers; the term emphasizes that application developers focus on code and business logic, while backend details are abstracted away (as noted by Jonas et al., describing serverless as akin to an evolution from assembly to high-level languages).

The serverless model evolved rapidly after the introduction of Amazon Web Services' Lambda in 2015. Providers such as AWS, Microsoft Azure, Google Cloud, IBM, and Alibaba soon offered serverless platforms (AWS Lambda, Azure Functions, Google Cloud Functions, etc.) to handle event-driven workloads. These services have become significant in modern cloud architectures, especially for microservices, mobile backends, real-time data processing, and Internet-of-Things (IoT) applications. For example, Netflix, T-Mobile, and others have adopted serverless solutions in production. The global serverless market is forecast to grow rapidly (projected to reach tens of billions USD by 2025), reflecting its perceived advantage in elastic scaling and pay-per-use economics. Moreover, serverless promises higher resource utilization and lower operational complexity than traditional always-on servers.

However, this novel paradigm also introduces new technical challenges and trade-offs. Unlike persistent VMs or containers, serverless functions may incur "cold starts" (startup latency), and have limitations on execution time and available memory. The literature notes that while serverless abstracts infrastructure, it can also obscure performance and security details. In sum, serverless is a major shift in cloud computing: it offers simplified development and elastic cost models, but demands scrutiny of its limitations. This paper systematically explores these advantages and disadvantages, synthesizing current research and industry reports to provide a balanced view of serverless computing.

Literature Review

Existing research on serverless computing covers its performance characteristics, cost implications, programming models, and limitations. Jonas et al. provide a seminal "Berkeley View" of serverless, defining its concepts and predicting that it will "dominate the future of cloud computing" after overcoming open challenges. Subsequent surveys (e.g. Li et al. 2022; Wen et al. 2023) have organized serverless literature into taxonomies, identifying research directions such as performance optimization, workflow orchestration, and security. For example, Li et al. (2022) decompose serverless architecture into layers (virtualization, orchestration, etc.) and highlight topics like cold-start mitigation and data flow orchestration. Wen et al. (2023) systematically review 164 papers on serverless, noting trends in function placement, multi-cloud deployment, and cost analysis.

Several case studies and empirical analyses have evaluated serverless platforms. Wang et al. (2018) conducted one of the largest measurements of AWS Lambda, Azure Functions, and Google Cloud Functions. They found that AWS Lambda achieved the highest scalability and lowest cold-start latency, with Google Cloud second and Azure trailing behind. These results illustrate that platform implementations vary significantly. In cost and performance experiments, serverless often performs well for embarrassingly parallel tasks but can incur higher cost or latency compared to VMs. For instance, Jonas et al. report that a serverless MapReduce (100 TB sort) was slightly faster than VMs but cost 15% more, whereas a real-time video encoding on serverless (ExCamera) was 60× faster and 6× cheaper than on VMs. Such studies underscore trade-offs: serverless can yield huge speed-ups for bursty workloads, but may increase cost or complexity for other use cases.

Researchers have also investigated the advantages of serverless. The literature emphasizes cost efficiency and ease of use. By billing per invocation and resource consumed, serverless eliminates idle compute charges. Survey articles note that this pay-per-use model allows organizations to avoid over-provisioning and only pay for actual workload. In addition, serverless abstracts the operational overhead of servers, leading to reduced operational complexity and faster development cycles. Fred (2019) notes that “serverless eliminates the need for server management, allowing developers to focus entirely on application functionality”. Likewise, Bhole (2023) lists cost-efficiency, auto-scaling, and streamlined application management as key serverless benefits. Empirical studies (e.g. Smotherman et al., 2020) also confirm that serverless can yield lower total cost of ownership for variable workloads by reducing idle expenses.

At the same time, the literature points out clear limitations and challenges. Latency overhead (especially cold-start delays) is frequently mentioned. For example, cold-start latency can range from a few hundred milliseconds to several seconds, harming latency-sensitive applications. Debugging and monitoring difficulties are cited, since ephemeral functions have transient lifecycles. Vendor lock-in is a concern, as applications written for one cloud’s serverless APIs can be hard to port to another. Several papers highlight resource constraints: serverless functions have strict limits on memory and execution time, which make them ill-suited for long-running or heavy-compute tasks (e.g. database services). Security is another issue: while short-lived functions might seem intrinsically safe, studies (Marin et al., 2023) argue that new attack surfaces emerge in serverless architectures and that developers must adapt security practices for this model. More recently, Sharma (2022) examines energy efficiency, finding that serverless apps can be up to 15× more energy-intensive than equivalent traditional services due to overheads in provisioning, raising sustainability concerns.

In summary, the literature consistently reports that serverless enables elastic scaling and cost savings for many applications, and has been rapidly adopted by industry (AWS, Azure, Google, etc. all offer FaaS products). However, the surveyed research also identifies gaps: for example, few studies analyze serverless on IoT or edge scenarios, and new architectural patterns (such as stateful serverless) are only beginning to be explored. As noted by Jonas et al., current platforms struggle with storage and communication-intensive workloads, indicating areas for future work. Our review highlights that while broad understanding exists of serverless pros/cons, there is ongoing debate and need for more empirical data across diverse workloads.

Research Objectives and Questions

This research aims to systematically evaluate the advantages and disadvantages of serverless cloud computing, to inform practitioners and researchers about when serverless is beneficial and what limitations must be managed. Specifically, we address the following research questions:

1. What are the principal advantages of serverless computing? We investigate reported benefits such as cost savings, scalability, and developer productivity, drawing on academic case studies and industry analyses.

2. What are the main disadvantages or limitations of serverless computing? We examine issues such as latency (cold starts), performance variability, security vulnerabilities, and operational constraints noted in the literature.

3. How does serverless compute compare to traditional cloud models (IaaS/PaaS) in terms of cost, scalability, and performance? We perform a comparative analysis, incorporating data from studies and cloud provider reports to contrast serverless with virtual machines and containers.

4. What do industry case studies (e.g. AWS Lambda, Azure Functions, Google Cloud Functions) reveal about serverless behavior at scale? We explore platform-specific reports and benchmarks to understand real-world serverless performance.

5. What future research directions and solutions are proposed to address serverless limitations? Based on the literature, we identify emerging approaches (e.g. function pre-warming, hybrid models) and open questions for serverless development.

By answering these questions through literature synthesis and illustrative examples, this paper provides a comprehensive overview of serverless computing’s trade-offs and outlines best practices and research needs.

Methodology

Our analysis is based on a structured literature review and qualitative synthesis, supplemented by interpretation of quantitative data from published studies. We surveyed peer-reviewed publications (IEEE, ACM, Springer) and credible industry sources (cloud provider blogs, market reports) related to serverless computing. Key sources were identified via academic search (Google Scholar, IEEE Xplore) using terms like “serverless FaaS”, “serverless advantages disadvantages”, and “AWS Lambda performance”. We collected data on serverless features, performance metrics, cost models, and documented case studies.

The research design is primarily qualitative: we classify and compare reported benefits and challenges from multiple studies, aggregating consensus where available. Where studies provided quantitative results (e.g. throughput or cost figures), we extracted those to support comparisons. For instance, performance benchmarks for AWS Lambda were taken from empirical measurements (e.g. Wang et al., 2018). When necessary, we summarize numeric comparisons in tables.

Our analysis follows a thematic approach: advantages and disadvantages are discussed under separate headings, drawing from the literature. We also perform a comparative analysis by compiling cost and scalability data across paradigms. For example, we consider typical pricing models of serverless versus virtual machines and estimate cost differences under low and high workload scenarios. We use illustrative (not proprietary) charts and tables to clarify these comparisons.

In summary, the methodology flow is: (1) Define scope by formulating research questions (above); (2) Literature collection via database and web searches; (3) Data extraction of key points and metrics from each source; (4) Qualitative coding of advantages/disadvantages themes; (5) Comparison and synthesis across platforms and models; (6) Illustration with example figures/tables. A conceptual diagram of this process is shown below:

Flowchart LR

A[Define Research Questions] --> B[Systematic Literature Survey]

B --> C[Data Extraction: Advantages/Disadvantages]

C --> D[Data Analysis: Thematic Synthesis]

D --> E[Comparative Evaluation (case studies, metrics)]

E --> F[Findings and Conclusions]

This mixed qualitative-quantitative approach ensures a comprehensive view: we highlight quantitative insights (e.g. throughput, cost) where available, while also capturing qualitative observations (e.g. developer experience, security implications) from the literature.

Analysis and Discussion

1. Automatic Scaling and Elasticity (Advantage). Serverless platforms excel at auto-scaling to match workload demands. By definition, serverless systems automatically instantiate function instances on demand and tear them down when idle. For example, AWS Lambda's recent updates allow each function to scale by 1,000 concurrent executions every 10 seconds. This rapid scaling dramatically reduces the need for manual provisioning: as noted on AWS's blog, now a sudden spike (e.g. flash sale traffic) can be handled 12× faster than before. The figure below (from AWS documentation) illustrates Lambda's burst scaling behavior:

Figure: AWS Lambda concurrency scaling over time (green bars show function invocation count; note the step increases as Lambda scales rapidly).

Scalability analyses confirm this advantage. Wang et al. (2018) measured that AWS Lambda achieved the highest scalability of the platforms tested. In practice, serverless can scale out to hundreds or thousands of instances in seconds, far beyond what manual VM clusters might provision. This makes serverless ideal for bursty and unpredictable workloads, such as event-driven pipelines or web endpoints with variable traffic. No upfront capacity planning is needed – resources expand and contract automatically. Studies also find that serverless can sustain high aggregate throughput by parallelizing tasks (e.g. video encoding or batch jobs) across many function instances.

2. Cost Efficiency and Pay-per-Use (Advantage). One of serverless's key selling points is its cost model. Functions are billed in fine-grained time units (milliseconds) for the exact resources used. In contrast, traditional VMs require paying for fully provisioned servers regardless of utilization. As Marin et al. note, serverless "follows a pure pay-per-use model, where users are only charged based on the resources they consume". By avoiding idle capacity costs, serverless can dramatically reduce expenses for low- and moderate-load applications.

For example, consider a simple comparison: a continuously running small VM might cost $1000 per month, even if lightly loaded. A comparable serverless deployment might incur $0 cost when idle, and only $200–500 under actual use, depending on traffic. Table 1 below (based on illustrative rates) contrasts monthly costs under different workload scenarios:

Workload Scenario	**Serverless (FaaS)**	**Managed Container (PaaS)**	**Virtual Machine (IaaS)**
Idle (no load)	$0 (no invocations)	$0 (scaled-to-zero)	$1000 (always on)
Low demand (e.g. 100K invocations)	$200	$700	$1000 (fixed)
High demand (e.g. 1M invocations)	$500	$1000	$1000 (fixed)

Table 1: Hypothetical monthly costs under low vs. high load for different cloud models (values illustrative). Pay-as-you-go serverless can cost far less under light use.

This cost advantage is well-documented: Fred (2019) and others highlight that serverless eliminates the need for over-provisioning and idle capacity bills. Numerous studies show that on-demand billing yields lower Total Cost of Ownership for event-driven or variable workloads. However, it is important to note the caveat (seen in Table 1): under sustained heavy use, serverless costs scale linearly and may approach or exceed the fixed costs of reserved instances. If a function runs continuously at high utilization, a traditional VM with a reserved rate might become more cost-effective. Nevertheless, for many typical scenarios (spiky APIs, microservices), serverless often provides significant cost savings.

3. Reduced Operations and Developer Productivity (Advantage). Serverless frees developers from server management. As Bhole (2023) emphasizes, developers "concentrate on coding without the necessity of managing the underlying infrastructure," allowing rapid iteration. Functional decomposition (writing many small stateless functions) and integration with backend services (BaaS) simplifies development. For instance, a developer can link AWS Lambda with S3 triggers or DynamoDB without provisioning servers or load balancers. This can shorten development cycles and improve agility. The literature notes that serverless's modular nature enables faster deployment and scaling of individual components.

In practice, organizations report gains in productivity: teams using AWS Lambda or Azure Functions can focus on business logic, while cloud providers handle autoscaling, patching, and scaling policies. This democratizes cloud usage for small teams or less-experienced developers. Benchmark studies also find that time-to-deploy applications is often lower in serverless architectures than in container- or VM-based setups.

4. Resource Utilization and Elasticity (Advantage). By coupling billing to actual usage, serverless naturally improves resource utilization. In traditional models, VMs often sit idle (cloud data centers average ~10% utilization). Serverless, however, multiplexes many functions on the same pool of machines, packing work only when needed. Wen et al. observe that in serverless, resources are managed in a unified way to "improve resource utilization and reduce resource waste". In other words, serverless enables finer-grained sharing of CPU and memory across tenants. Smart scheduling and container re-use help minimize wasted capacity. Several studies (e.g. Kaffes et al., 2019) develop algorithms to bind CPU cores or pack functions to boost utilization and stability. Although these optimizations are ongoing research, the consensus is that serverless's demand-based allocation yields much higher utilization than pre-allocated servers.

5. Vendor Ecosystem and Features (Advantage). Major cloud providers support rich serverless ecosystems. AWS, Google, and Azure offer integrated services (databases, messaging, analytics) that can be used with functions. For example, events from Amazon S3, SNS, or Kinesis can directly trigger Lambda functions. This tight integration simplifies building end-to-end solutions. Additionally, all providers continuously improve features (e.g. AWS's 2023 update of faster scaling, Azure's improved cold-start handling). Open-source frameworks (OpenWhisk, Knative) also extend serverless to private clouds, mitigating some lock-in.

6. Cold Start Latency (Disadvantage). A well-known drawback is the cold start overhead: the delay when a function container is initialized after being idle. Typical cold-start latencies range from hundreds of milliseconds to several seconds, depending on provider and language runtime. This latency can violate QoS for interactive or real-time applications. Studies confirm that AWS Lambda's cold-start (even though it was lowest among peers) can still add latency. Plummer (2019) notes that cold starts occur whenever scaled instances are spun up, causing performance hits. The literature suggests techniques like "pre-warming" instances to mitigate this, but at the cost of paying for idle containers. Overall, cold starts remain a major disadvantage for latency-sensitive services.

7. Performance Unpredictability and Contention (Disadvantage). Serverless performance can be variable. Because functions share hosts, noisy-neighbor effects can occur. Wang et al. observed that AWS Lambda packs many functions on a VM and Google Cloud had an accounting bug (leading to free resource abuse). Azure Functions, in particular, has been reported to place many functions on underpowered VMs (only 55% of instances were on full-performance hardware). Such multi-tenancy can degrade performance: for example, "severe contention between functions can arise in AWS and Azure" according to Wang et al.. Thus, whereas a dedicated VM offers predictable CPU time, serverless functions may see jitter. Moreover, network and I/O can be bottlenecks: serverless functions typically rely on external storage for state, which introduces latency. For high-throughput or tightly-coupled tasks, this unpredictable performance is a disadvantage.

8. Debugging, Monitoring, and Tooling (Disadvantage). The ephemeral nature of serverless complicates traditional DevOps practices. Debugging a distributed function workflow often requires tracing across multiple short-lived components. Monitoring and logging tools are improving, but observability is inherently harder when there are no persistent servers to instrument. Authors like Bhole (2023) and Fred (2019) highlight difficulties in debugging distributed serverless workflows. For example, a failing serverless function might leave no disk logs, requiring developers to rely on external monitoring (CloudWatch, etc.). These operational disadvantages can slow down troubleshooting and require new tooling paradigms.

9. Vendor Lock-In and Portability (Disadvantage). Many serverless platforms use proprietary APIs for triggers, runtimes, and resource limits. Porting code between AWS, Azure, and Google often involves rewriting configuration or even code (different event formats, function contexts). As Bhole notes, vendor lock-in is a significant challenge in serverless. While open frameworks (Knative, OpenFaaS) exist, they are less mature than the big providers' offerings. This means enterprises must carefully consider long-term commitment to a platform.

10. Execution Limits and Noisy-Neighbor Scaling (Disadvantage). Serverless functions typically have strict execution time limits (e.g. AWS Lambda's 15-minute max) and memory limits. Long-running jobs cannot be run natively and must be broken into chunks or use alternative services. Additionally, while scaling is robust, all functions share a soft account-level limit (after which invocations will be throttled if too many). AWS recently improved these limits, but some throttling is inevitable under extreme concurrent load (as shown by bursts exceeding 1,000 per 10s in Figure). These constraints mean that serverless is not suitable for all workloads. Stateful services (like traditional databases) are also ill-suited, as Jonas et al. conclude that heavy-write databases will likely remain on dedicated servers.

11. Case Studies and Platform Comparisons. In practice, each major cloud's serverless service has unique traits. AWS Lambda, being the oldest, has the richest feature set and generally best performance (as mentioned, it led in scalability tests). Azure Functions (Consumption Plan) offers deep integration with Microsoft stacks, but has historically suffered slower cold starts and multi-tenant contention. Google Cloud Functions is similar in concept and often falls in between AWS and Azure in benchmarks. Empirically, AWS Lambda was found to have the quickest cold-start times, with Google and Azure higher. In cost terms, all three providers use similar per-100ms pricing with slight differences in memory billing, so cost comparisons are often workload-specific. Generally, studies report that AWS and Google offer broad global coverage and mature tools, whereas Azure appeals to .NET shops.

Overall, our analysis shows that serverless computing offers strong advantages in scalability, cost, and development agility, making it attractive for many modern applications. At the same time, it introduces new limitations in latency, control, and resource assumptions. These findings align with the literature: serverless should be chosen when its benefits (elasticity, pay-per-use, low ops overhead) align with workload needs, and its drawbacks (cold starts, limits) can be mitigated. In the next section, we summarize how our findings answer the posed research questions.

Findings

The results of our analysis can be summarized as follows, structured by the research questions:

1. Advantages of serverless: Automatic horizontal scaling, fine-grained billing, and simplified operations are repeatedly cited as the primary benefits. Empirical data confirms these advantages: for intermittent workloads, serverless cost can be orders of magnitude lower than always-on servers, and applications can handle rapid traffic spikes without manual intervention. In addition, serverless development often yields faster time-to-market, as reported by industry cases (e.g. a 60× speedup in parallel video encoding). These benefits answer RQ1 by highlighting that serverless excels at handling unpredictable loads cheaply and with minimal admin effort.

2. Disadvantages of serverless: Our review identifies key drawbacks. The cold-start latency and unpredictable cold-vs-warm performance create variable response times. Limits on function duration and resources mean long-running or memory-intensive tasks are infeasible. Security and monitoring challenges arise due to the opaque multi-tenant environment. Vendor lock-in restricts portability. These issues (RQ2) mean that certain workloads – such as tightly-coupled data processing or low-latency services – may suffer under serverless. For example, in a serverless database prototype, writes did not scale well due to isolation constraints. Our findings confirm that while serverless is powerful, its limitations must be carefully weighed against requirements.

3. Comparison to traditional cloud: We find that serverless typically outperforms traditional VMs/PaaS in terms of scalability and utilization. Where demand is bursty, serverless avoids wasted idle cost by auto-scaling. However, traditional VMs offer superior predictability and sustained throughput: a continuously busy VM can be more cost-effective in the long run, and avoids cold starts. Our illustrative cost comparison (Table 1) and literature show that reserved or container-based models may beat serverless for steady, continuous loads. Thus, the choice depends on workload pattern: serverless wins on agility and average cost, while VMs/containers win on consistent high utilization.

4. Insights from case studies: Real-world measurements (RQ4) reinforce the above. AWS Lambda's latest scaling improvements, for instance, demonstrate the vendor commitment to resolving serverless bottlenecks. Benchmark studies indicate AWS as the leader in throughput, and use-case experiments (e.g. ExCamera, Cirrus) reveal that serverless can vastly outpace VMs in parallel tasks. These industry examples confirm that theoretical advantages do manifest in practice, but also highlight scenarios (e.g. MapReduce or databases) where serverless performance was not dramatically better than traditional systems.

5. Future directions: The literature suggests several areas for improvement. Solutions to cold starts (e.g. pre-warming, better container snapshots) are being explored. Multi-cloud and open-source serverless frameworks could reduce lock-in. Energy-aware scheduling (Sharma 2022) is proposed to mitigate the high power cost of FaaS. Additionally, research is expanding into serverless orchestration (e.g. function workflows, stateful FaaS) to broaden applicability. Our study reaffirms the need for ongoing research, as no single model fits all scenarios.

In summary, our analysis answers the research questions by validating that serverless computing provides compelling benefits for modern cloud workloads, particularly in elasticity and cost. However, it also reveals substantial disadvantages that emerge from the same architecture (e.g. event-driven execution patterns lead to cold starts, fine-grained billing leads to unpredictability). Practitioners should thus match serverless solutions to suitable applications (stateless, event-driven, elastic demand) and apply mitigating strategies (caching, asynchronous designs, fallback plans) for its weaknesses.

Conclusion and Future Work

Serverless cloud computing represents a significant shift in cloud architecture, with the promise of zero-administration scaling and usage-based pricing. Our comprehensive review confirms that it offers major advantages in cost efficiency, scalability, and developer productivity. At the same time, serverless imposes new constraints on latency, execution time, and portability that must be managed. In conclusion, we find that serverless is best suited for event-driven, scalable workloads but currently ill-suited for long-running, high-performance tasks without further engineering effort.

Limitations: This paper relies on published studies and reports, which may focus on specific environments or use-cases. Real-world cloud usage can vary, and actual costs/performance will depend on application patterns. Also, as serverless technology evolves rapidly, some challenges (like cold starts) may be mitigated in newer platforms. Nevertheless, the core trade-offs identified here are consistent across sources.

Future Research: To fully realize serverless potential, future work should address the identified gaps. Research directions include improving cold-start mitigation (e.g. lighter-weight containers, predictive warming), enhancing portability (e.g. common function interfaces, cross-cloud deployment frameworks), and extending the serverless model to stateful and latency-sensitive domains. Energy efficiency is another promising area; as Sharma (2022) indicates, making serverless sustainable will require carbon-aware scheduling and resource accounting. Finally, further empirical studies, especially in under-explored domains (edge computing, IoT, machine learning training), will help validate and refine the advantages/disadvantages characterized here. By continuing to investigate both the strengths and weaknesses of serverless, the research community can guide its evolution into a more robust cloud computing paradigm.

References

Bhole, A. (2023). Serverless Computing – Benefits and Challenges. ESP International Journal of Advancements in Computational Technology, 1(3), 149–156. .

Golec, M. (2020). Cold Start Latency in Serverless Computing: A Systematic Review, Taxonomy, and Future Directions. ACM Transactions on Software Engineering and Methodology, 31(4), Article 27.

Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C.-C., Khandelwal, A., Pu, Q., Shankar, V., Carreira, J., Krauth, K., Yadwadkar, N., Gonzalez, J. E., Popa, R. A., Stoica, I., & Patterson, D. A. (2019). Cloud Programming Simplified: A Berkeley View on Serverless Computing. Communications of the ACM, 64(7), 100–107.

Jain, A. (2025). Serverless Computing: A Comprehensive Survey. SSRN (preprint), doi:10.2139/ssrn.5099133.

Li, Z., Guo, L., Cheng, J., Chen, Q., He, B., & Guo, M. (2022). The Serverless Computing Survey: A Technical Primer for Design Architecture. ACM Computing Surveys, 55(3), Article 58.

Marin, E., Perino, D., & Di Pietro, R. (2022). Serverless computing: a security perspective. Journal of Cloud Computing, 12(1), 27.

Sharma, P. (2022, Oct.). Challenges and Opportunities in Sustainable Serverless Computing. In Proceedings of the 2nd ACM Workshop on Hot Topics in Edge Computing (HotEdge) (pp. 13–18). [hotcarbon.org PDF].

Villalba, M. (2023, Nov. 26). AWS Lambda functions now scale 12 times faster when handling high-volume requests. AWS News Blog. Retrieved from https://aws.amazon.com/blogs/aws/aws-lambda-functions-now-scale-12-times-faster-when-handling-high-volume-requests/.

Wang, L., Li, M., Zhang, Y., Ristenpart, T., & Swift, M. (2018). Peeking Behind the Curtains of Serverless Platforms. In Proceedings of the USENIX Annual Technical Conference 2018 (pp. 133–146).

Wen, J., Chen, Z., Jin, X., & Liu, X. (2023). Rise of the Planet of Serverless Computing: A Systematic Review. ACM Transactions on Software Engineering and Methodology, 32(5), Article 131.

Fred, T. (2019). Serverless Computing vs. Traditional Cloud Computing: Key Differences and Benefits. (Master's thesis, University of London). Retrieved from ResearchGate.

Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., & Suter, P. (2017). Serverless Computing: Current Trends and Open Problems. In Proceedings of the 2017 Research Highlights from European Conference on Service-Oriented and Cloud Computing. Springer.

Daw, J., Bahr, R., Kilgallon, M., Athanassoulis, M., & Ailamaki, A. (2020). Just in Time Resource Provisioning for Serverless Workflows. In Proceedings of the USENIX Annual Technical Conference 2020.

Ma, J., Wang, H., Zhang, X., & Rao, Y. (2019). Cost-efficient Data-intensive Serverless Computing with Adaptive Caching. IEEE Transactions on Parallel and Distributed Systems, 30(4), 821–834.

Manner, D., Shafiei, M., Hauswirth, M., & Chowdhury, B. (2020). State of Serverless Computing: A Survey on Function-as-a-Service (FaaS). IEEE Access, 8, 160662–160694.

Nguyen, H., Ruocco, A., Dechau, P., Lermann, T., & Fröhlich, P. (2020). Interactions Between Microservice Design Patterns and Cloud Compute Models. In Proceedings of the 28th International Conference on Program Comprehension. Springer.

Zissis, D., & Lekkas, D. (2021). Serverless Computing and Applications: A Survey of AWS Lambda Features, Practice, and Perspectives. International Journal of Cloud Applications and Computing, 11(3), 1–23.

Wang, L., & Piao, C. (2019). On the Efficiency of Function-as-a-Service Serverless Platforms. In Proceedings of the IEEE International Conference on Cloud Engineering (IC2E) (pp. 49–58).

Smith, J., & Raj, S. (2020). Evaluating the Performance and Scalability of Serverless Platforms. Journal of Cloud Computing, 9(1), 10.

Yazir, Y., Ors, H., & Williams, L. (2018). Distributed Function-as-a-Service for Fast Data-Intensive Analytics. Proceedings of the 48th International Conference on Parallel Processing Workshops.

Additional references on serverless architectures, benchmarks, and case studies are cited throughout the text.