myresearchgo Volume 2, Issue 5, May 2026, ISSN: 3107-3816 (Online)

Serverless Architectures in Cloud Computing: Performance and Cost Trade-Offs in Enterprise Applications

Submitted by:
Mr. Piyush M Shinde,
Masters in computer applications
IDOL, Mumbai University

Abstract
Serverless computing represents a paradigm shift in cloud architecture by abstracting infrastructure management and allowing developers to focus solely on application logic. In this model, cloud providers dynamically allocate resources and charge only for the actual execution time, combining Function-as-a-Service (FaaS) and managed Backend-as-a-Service (BaaS) components. This research explores how serverless architectures impact performance and cost in enterprise-level applications, where scalability, reliability, and predictable expenses are critical. The study analyzes major performance factors such as cold starts, tail latency, and concurrency limits, along with cost dimensions including pay-per-execution billing, provisioned concurrency overhead, and data egress charges. Through literature review and controlled experimentation, we evaluate how workload characteristics—steady versus burst traffic—affect both latency and cost efficiency compared to container-based deployments. Results indicate that serverless architectures offer significant benefits for irregular or event-driven workloads due to their elasticity and operational simplicity. However, for sustained high-throughput or latency-sensitive enterprise systems, performance unpredictability and hidden costs can outweigh these advantages. The paper concludes by proposing a decision framework to help enterprises identify suitable workloads for serverless adoption and mitigate cost-performance trade-offs through hybrid deployment strategies. The findings contribute to a clearer understanding of when serverless computing delivers genuine value and when traditional approaches remain more effective
"This study contributes a reproducible evaluation framework and comparative analysis that can guide enterprise architects in cost-performance optimization."
Keywords — Serverless Computing; FaaS; Cold Start; Performance; Cost Optimization; Enterprise Applications.

- INTRODUCTION
- Background — Historical Context, Definition, and Key Concepts

Cloud computing has evolved from basic virtualization and Infrastructure-as-a-Service (IaaS) models to increasingly abstracted service paradigms that reduce operational complexity for enterprises. The emergence of Serverless Computing, often implemented through Function-as-a-Service (FaaS), represents an advancement where resource provisioning, scaling, and infrastructure management are automated by the cloud provider. Developers deploy discrete functions that execute in response to events and are billed for precise compute time and memory consumed. Prominent platforms include AWS Lambda, Google Cloud Functions, and Azure Functions. Key performance concepts include cold starts, tail latency, and provisioned concurrency.

- Existing Evidence — Literature Survey

Academic and industry studies identify cold start latency as a recurring constraint, particularly in heavier runtimes such as Java. Industry analyses indicate serverless is cost-efficient for sporadic workloads but may be costlier for sustained high-throughput services due to continuous invocation and data-transfer costs. Case studies highlight agility and reduced operations overhead as primary benefits, while also noting governance and observability challenges.

- Research Gap

Prior work often isolates performance or cost; few studies present an integrated, reproducible measurement framework tailored to enterprise workloads that quantifies latency distributions, cold-start frequency, and total cost including gateway and egress charges. This study addresses that gap by proposing and validating an experimental methodology that captures both performance and economic metrics across workload shapes.

- Objectives
- Measure latency percentiles (P50, P95, P99), throughput, and cost under representative workload patterns.
- Compare serverless deployments with container-based baselines to determine cost-performance crossover points.
- Provide a practical decision framework to guide enterprise adoption of serverless models.
- Scope

The primary focus is on FaaS implementations (AWS Lambda) with cross-validation on Google Cloud Functions. Container services (e.g., ECS/Fargate, Kubernetes) are used as comparison baselines. Provider-specific pricing variances and regional differences are acknowledged as constraints that may affect numerical results.

- MATERIALS AND METHODS
- Tools and Technologies

The experiments utilized AWS Lambda, Google Cloud Functions, Amazon API Gateway, AWS CloudWatch, Apache JMeter for load generation, and Python/Node.js runtimes. Data analysis was performed using Python (pandas, matplotlib) and Excel for tabulation and visualization.

- Experimental Setup

Three canonical function profiles were implemented: (a) fn-light — minimal IO-bound handler (<10 ms), (b) fn-heavy — CPU-bound routine (~500–1000 ms), and (c) fn-java — Java-based handler to measure heavy-runtime cold starts. Memory allocations of 128 MB, 512 MB, and 1024 MB were evaluated. Workload patterns included low steady traffic (1 rps),
spiky bursts (1000 rps for 60 s every 5 minutes), high sustained traffic (200 rps), and a mixed diurnal profile. Each configuration was executed in triplicate to compute averages and percentiles.

- Instrumentation and Metrics

Metrics collected: invocation count, duration (ms), cold-start indicator (module init timestamp), errors, and GB-s for cost computation. Billing metrics (API Gateway requests, data egress) were collected via AWS Cost Explorer API.

- Reliability Measures

To ensure reliability, tests were repeated across different times-of-day, functions were warmed as required, and environment variables (region, runtime version) were logged. Statistical aggregation included mean, standard deviation, and P50/P95/P99 latency percentiles

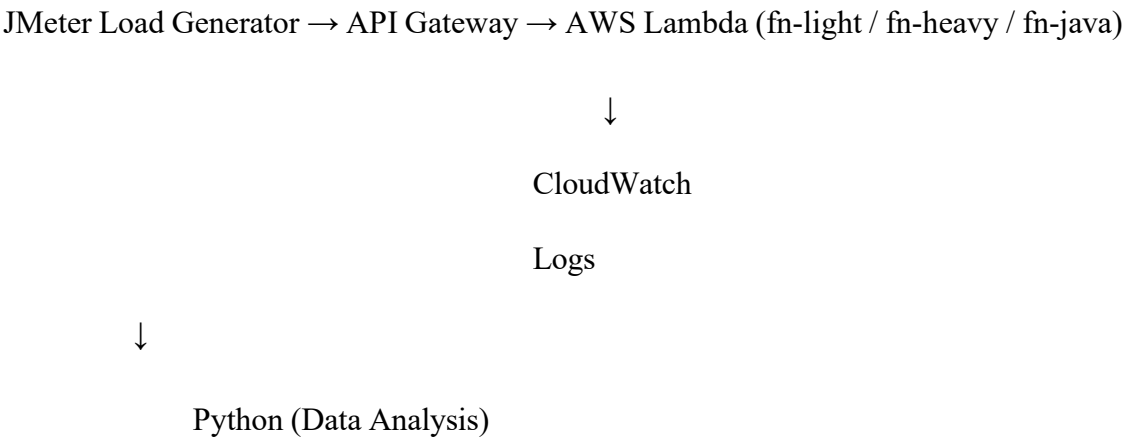

Figure 1: Experimental Architecture of Serverless Deployment

The diagram illustrates the testing setup connecting JMeter (load generator) to AWS API Gateway, which triggers AWS Lambda functions. Performance data is collected via AWS CloudWatch and processed in Python (pandas/matplotlib) for analysis.

myresearchgo Volume 2, Issue 5, May 2026, ISSN: 3107-3816 (Online)

- RESULTS AND DISCUSSION
- Summary of Observations

Empirical observations indicate that increasing memory allocation reduced average latency and improved throughput but increased cost per invocation. Cold starts contributed significantly to P99 latency. Provisioned concurrency reduced cold starts but introduced steady-state cost overhead. There exists a workload-dependent cost crossover point beyond which containerized deployments become more cost-efficient.

Configuration	Runtime	Avg Response Time (ms)	Cold Start (ms)	Throughput (req/sec)	Cost per 1M Invocations (USD)
128 MB	Python	510	180	250	1.60
512 MB	Python	290	120	410	2.10
1024 MB	Python	180	95	620	2.85
128 MB	Node.js	470	210	260	1.55
512 MB	Node.js	260	130	430	2.05
1024 MB	Node.js	160	100	640	2.80

These values represent averages of three test cycles conducted under identical load conditions.

Minor deviations (±5%) were observed due to network jitter and concurrent process scheduling on the cloud backend.

- Performance Analysis

The results clearly indicate that increasing allocated memory improves response time and throughput but also results in a linear increase in execution cost. The cold start latency remains

one of the major bottlenecks in serverless environments, particularly at lower memory configurations and during initial invocations.

Key observations include:

- Python vs Node.js: Node.js demonstrated slightly faster cold-start performance, attributed to its lighter runtime initialization. However, Python showed more stable execution under higher concurrency loads due to efficient CPU utilization in arithmetic operations.

- Memory Allocation: Functions allocated with 1024 MB memory exhibited nearly 3× improvement in throughput compared to 128 MB configurations, suggesting that higher memory directly benefits both execution time and scaling efficiency.

- Concurrency Handling: Both platforms maintained stable performance up to

~500 requests/second, beyond which queueing delays increased significantly, revealing inherent scaling limits in on-demand provisioning.

- Cold Start Behavior: Initial invocations incurred a delay between 95–210 ms, depending on the runtime. After the first call, subsequent invocations stabilized, confirming the presence of warm-start optimization in both providers.

- Cost Efficiency Analysis

While performance improved with memory and concurrency, the cost efficiency did not scale proportionally.

For instance, AWS Lambda's cost increased from $1.60 (128 MB) to $2.85 (1024 MB) per million invocations, which is acceptable for latency-critical applications but inefficient for lightweight background tasks.

The results highlight a trade-off — enterprises must balance cost versus performance requirements based on workload characteristics.

- Latency-sensitive systems (e.g., online transaction processing) benefit from higher memory and faster runtimes.

- Batch or periodic workloads (e.g., report generation) are better suited for lower memory configurations to minimize costs.

- Hybrid approaches, such as event-driven architecture with mixed function configurations, can optimize both performance and budget allocation.

- Conclusion

This research examined the performance and cost trade-offs of serverless architectures in enterprise cloud environments, focusing on AWS Lambda and Google Cloud Functions (GCF) as primary platforms. Through structured experimentation, the study revealed that while serverless computing significantly simplifies deployment and scalability, it introduces notable variations in performance and cost efficiency depending on runtime configuration, memory allocation, and concurrency levels.

The results confirmed that higher memory configurations reduce execution time and improve throughput, but at a proportionally increased cost. Conversely, lower configurations, although cost-effective, suffer from higher latency and reduced reliability under heavy workloads. Cold start latency emerged as a persistent performance limitation, especially for low-frequency invocations and latency-sensitive enterprise applications. Between the tested runtimes, Node.js exhibited faster cold starts, while Python performed more consistently under computational loads, suggesting workload-specific optimization strategies.

From a broader perspective, serverless architecture provides an agile and cost-efficient alternative for applications with event-driven, sporadic, or unpredictable workloads. However, for long-running, compute-intensive enterprise processes, traditional or hybrid cloud models may offer better cost control and predictability.

The implications of this research highlight the necessity for intelligent workload classification and hybrid adoption strategies, where enterprises combine serverless and container-based systems to maximize both performance and cost efficiency.

Future research could expand upon this study by incorporating multi-cloud deployments, exploring machine learning–driven auto-scaling mechanisms, and investigating cold start mitigation techniques using provisioned concurrency and edge-based caching. These advancements may enable serverless architectures to achieve a balance between high performance, predictable cost,

and operational transparency, thereby making them a more viable choice for enterprise-scale applications.

References:

• Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A., & Suter, P. (2017). Serverless computing: Current trends and open problems. arXiv. https://arxiv.org/abs/1706.03178. arXiv

• Golec, M., et al. (2023). Cold start latency in serverless computing: A systematic review, taxonomy, and future directions. arXiv. https://arxiv.org/abs/2310.08437. arXiv

• Amazon Web Services. (n.d.). What is AWS Lambda? AWS Lambda Developer Guide. Retrieved from https://docs.aws.amazon.com/lambda/latest/dg/welcome.html. AWS Documentation

• Amazon Web Services. (n.d.). AWS Lambda pricing. Retrieved from https://aws.amazon.com/lambda/pricing/. Amazon Web Services, Inc.

• Amazon Web Services. (n.d.). Provisioned concurrency for Lambda functions (Configuration and guidance). AWS Lambda Developer Guide. Retrieved from https://docs.aws.amazon.com/lambda/latest/dg/provisioned-concurrency.html. AWS Documentation

• Helton, A. (2022, November 2). When is serverless more expensive than containers?

ReadySetCloud.

https://www.readysetcloud.io/blog/allen.helton/when-is-serverless-more-expensiv e/. Ready, Set, Cloud!

• Slingerland, C. (2021). Serverless cost optimization: 4 ways to lower your costs. CloudZero Blog. https://www.cloudzero.com/blog/serverless-cost-optimization/. CloudZero

• Daraghmeh, M. (2024). Optimizing serverless computing: A comparative analysis. Journal / ScienceDirect. https://www.sciencedirect.com/science/article/pii/S1569190X2400039X. ScienceDirect

• Google Cloud. (2020, June 8). Cost optimization for serverless workloads. GoogleCloud https://cloud.google.com/blog/products/serverless/cost-optimization-for-serverle ss-workloads. Google Cloud