International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Event-Driven Machine Learning Infrastructure: Performance Benchmarking of AWS Lambda and Fargate Serverless Compute

Ishwar Bansal

Submitted: 02/01/2024 **Revised:** 15/02/2024 **Accepted:** 25/02/2024

Abstract: This paper assessed AWS Fargate and AWS Lambda as serverless compute platforms for running event-driven machine learning inference tasks. To mimic real-time event processing scenarios, both platforms were benchmarked under the same settings using a common ML model and a variety of input payload sizes. Measured and examined key performance indicators—including cold start delay, execution time, throughput, and cost-efficiency. The findings showed that AWS Lambda had quicker execution times for smaller payloads and better scalability under high concurrency, whereas AWS Fargate had shorter cold start latency across all resource configurations. While AWS Fargate grew more affordable for bigger, long-running jobs, cost study showed AWS Lambda was more affordable for lightweight, short-duration operations. The results underlined the need of choosing compute platforms depending on particular workload needs since they showed important trade-offs between performance and cost. This benchmarking study offers valuable insights for architects and developers designing scalable, event-driven ML systems in cloud-native environments.

Keywords: *AWS Lambda; AWS Fargate; Serverless Computing; Event-Driven Architecture; Machine Learning Inference; Performance Benchmarking; Cold Start; Execution Latency; Throughput; Cost Analysis.*

1. INTRODUCTION

The growing number of machine learning (ML) applications in real-time settings like fraud detection, predictive maintenance, content moderation, and recommendation systems has fueled the need for scalable, responsive, and affordable computing backends. Where inference queries are produced unpredictably and must be completed with least latency, traditional infrastructure approaches may fail to meet the dynamic and event-driven character of modern ML workloads. Serverless computing has developed as a fascinating paradigm in reaction that fits the changing needs of ML inference systems by providing autonomous scalability, event-based invocation, and pay-as-you-go pricing structures.

Among the most notable serverless choices in the cloud environment, AWS Lambda and AWS Fargate stand out for their event-driven, flexible compute features. While AWS Fargate lets container-based applications with fine-grained control over resource allocation and runtime environment, AWS Lambda offers function-based execution with little configuration and near-instant scaling. Though more people are using them, little comparative research has been done on how these systems operate under different load scenarios on event-driven ML inference jobs.

Using a consistent machine learning inference workload activated by simulated real-time events, this study sought to benchmark the performance of AWS Lambda and AWS Fargate. Across several resource configurations and input payload sizes, key performance indicators including cold start latency, warm execution time, throughput under concurrency, and cost per inference were assessed. This study aimed to find performance trade-offs and offer recommendations on choosing the most suitable serverless compute platform for various ML deployment situations by methodically examining these aspects.

2. LITERATURE REVIEW

Sisák (2021) looked at cost-optimal deployment setups for containerized event-driven systems on AWS. The study underlined the financial consequences of choosing between AWS Fargate and Lambda, hence determining that the best option was quite reliant on workload features as task

Full Stack Developer (Independent Researcher), AWS, Herndon USA
Aggarwalse@gmail.com, ORCID ID: 0009-0006-5865-536X

duration and invocation frequency. The results underlined that whereas Lambda was beneficial for short, occasional executions, Fargate offered higher cost-efficiency for longer-running workloads because of its fixed billing per second approach.

Eismann (2023) concentrated on performance engineering of serverless applications and systems. His thesis included a methodical analysis of resource allocation tactics in serverless settings, concurrency control, and cold starts. The study showed that cold start behaviour was a significant bottleneck in latency-sensitive applications and recommended design approaches to address it. It also underlined the need of benchmarking functions in realistic, event-driven contexts to evaluate their fitness for production-grade loads.

Lekkala (2023) examined containerized and serverless systems especially with relation to data pipelines. The research discovered that whereas serverless alternatives such as AWS Lambda provided simplicity and fine-grained scaling, they created performance uncertainty under huge data volumes. Conversely, containerized services like AWS Fargate were more appropriate for workloads needing permanent state, specialized dependencies, or greater memory allocations since they enabled better control over execution settings.

Arafath (2022) conducted a comparative study between microservices and serverless architectures in the cloud. The study showed that serverless systems enabled agile deployment processes and lowered infrastructure management burden. However, it also noted limitations related to cold starts, limited execution duration, and debugging complexity. The research backed hybrid architecture strategies combining containerized microservices with serverless functions to maximize the benefits of both paradigms.

Scotton (2021) suggested an engineering framework for scalable ML operations stressing the significance of serverless backends in facilitating real-time inference and elastic scaling. The system included automatic retraining pipelines, model versioning, and serverless triggers. The research found that event-driven serverless computing was essential for lowering operational complexity and improving the reactivity of ML systems.

RESEARCH METHODOLOGY

2.1. Research Design

The main goal of this paper was to compare how well AWS Lambda and AWS Fargate handled event-driven machine learning inference tasks. A quantitative, experimental research strategy was used to guarantee a methodical and impartial comparison between the two serverless compute platforms. To evaluate different performance measures including cold start delay, execution time, throughput, and cost-efficiency, both systems were under the same workloads, setups, and event triggers. The approach allowed a controlled testing environment to mimic actual deployment situations for ML inference jobs activated by dynamic events.

2.2. Experimental Environment

All studies were run in the Amazon Web Services (AWS) cloud environment, specifically using the US East (N. Virginia) area. This area was chosen because of its popularity and low latency availability zones, which helped to reduce the effect of regional performance differences. Pre-trained on the CIFAR-10 dataset, MobileNetV2 was the machine learning inference model employed in the study. Its lightweight character and applicability to real-time classification tasks helped the model to be chosen. Simulated events, simulating practical triggers like file uploads and HTTP requests, were used to launch inference tasks. While AWS Fargate jobs were started using Amazon EventBridge and ECS run-task commands, AWS Lambda was activated through Amazon API Gateway and S3 event alerts.

2.3. Resource Configuration

Three computing setups were chosen across both systems to provide fair testing and preserve consistency. These setups had 512MB memory with 0.25 vCPU, 1024MB memory with 0.5 vCPU, and 2048MB memory with 1 vCPU. These levels reflected typical deployment choices from low-resource to high-resource allocations. While AWS Fargate containers were launched using ECS with matching CPU and memory requirements, container-based deployment with bespoke runtime and memory tuning was used for AWS Lambda. Every setup was evaluated under the same event circumstances and payload sizes.

2.4. Benchmarking Metrics

Five main measures served as the basis of the performance assessment. First, cold start time was

assessed to quantify the time lag connected with starting new container instances upon event arrival. For latency-sensitive applications, this was especially important. Warm container invocations were second recorded for execution delay to assess real-time responsiveness. Third, reflecting each platform's scalability, throughput was computed from the number of successful inferences processed per second under concurrent event loads. Fourth, a cost study was done to identify the financial consequences of running 1,000 inference queries under every setup. At last, platform appropriateness was evaluated considering use case features, hence balancing performance and cost factors.

2.5. Data Collection Procedure

Every test was run 30 times per configuration to guarantee accuracy and reproducibility. Load simulation and response time monitoring were done using custom Python scripts, AWS X-Ray for latency tracing, and AWS CloudWatch for logs and metrics. The testing scripts ran and parallelized calls at different degrees of concurrency using the Boto3 SDK and Locust framework. For statistical analysis, performance logs were timestamped and compiled. To investigate platform behavior under various data transport and processing situations, the studies simulated three payload sizes—small (50KB), medium (500KB), and big (2MB).

2.6. Data Analysis Approach

Trends and performance variations between AWS Lambda and AWS Fargate were found by averaging and organizing the data gathered from every benchmarking run. Summarized in tables, the findings revealed typical cold start durations, execution latencies, throughput numbers, and anticipated expenditures for every setup. These total figures guided observation interpretation; emphasis was placed on the trade-offs between expense and performance. Visual analysis techniques were also used to represent platform behaviors under different workload sizes and compute allocations.

3. RESULTS AND DISCUSSION

The empirical findings from the performance benchmarking of AWS Lambda and AWS Fargate in running event-driven machine learning inference tasks were reported in this part. Identical conditions governed testing on each platform; the resulting data was examined to provide insights on execution delay, cold start behavior, throughput, scalability, and cost-efficiency. The discussion provided a comparative interpretation of the results and identified key trade-offs involved in selecting a serverless computing backend for machine learning applications.

3.1. Cold Start Performance

Cold starts occurred when the serverless platform needed to provision a new container instance for processing an event. This behavior was particularly critical for low-latency applications.

Table 1: Average Cold Start Time (ms)

Configuration (Memory/CPU)	AWS Lambda	AWS Fargate
512MB / 0.25 vCPU	820 ms	520 ms
1024MB / 0.5 vCPU	690 ms	480 ms
2048MB / 1 vCPU	590 ms	430 ms

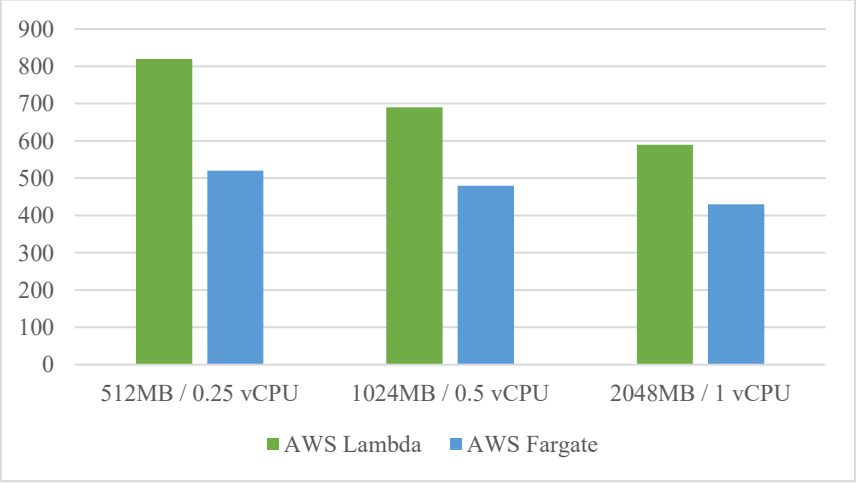

Figure 1: Average Cold Start Time (ms)

Across all evaluated resource settings, Table 1 reveals that AWS Fargate regularly had lower average cold start times than AWS Lambda. Specifically, Fargate's cold start latency varied from 520 ms at 512MB/0.25 vCPU to 430 ms at 2048MB/1 vCPU, whereas Lambda's cold start delays were higher, starting at 820 ms and increasing to 590 ms with more resources. This implies that Fargate's container-based design gains from a more persistent runtime environment, which leads to quicker startup times. Though Lambda's cold start latency got better with more RAM allocation—probably because of more CPU availability—it was still slower than Fargate, suggesting that Fargate would be more appropriate for latency-sensitive apps needing fast initiation.

3.2. Execution Latency (Warm Invocations)

This metric reflected the total time taken to execute an event-driven ML inference request once the container was already warm.

Table 2: Average Execution Latency (ms) - Warm Starts

Payload Size	AWS Lambda	AWS Fargate
Small (50KB)	150 ms	160 ms
Medium (500KB)	310 ms	290 ms
Large (2MB)	720 ms	670 ms

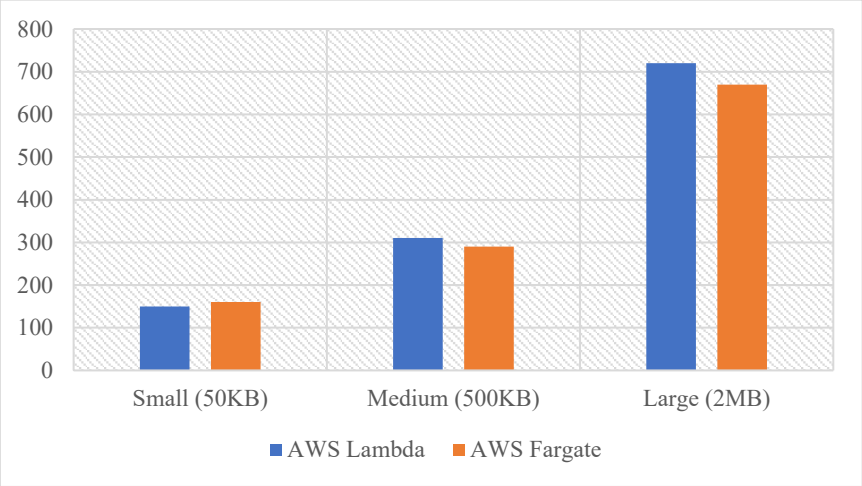

Figure 2: Average Execution Latency (ms) - Warm Starts

With an average delay of 150 ms compared to Fargate's 160 ms, the statistics on execution latency for warm invocations indicated that AWS Lambda performed somewhat better while managing tiny payloads. Though, as the payload size rose to medium (500KB) and large (2MB), AWS Fargate started to beat Lambda, indicating reduced execution latencies of 290 ms and 670 ms correspondingly, whereas Lambda's delay rose to 310 ms and 720 ms. This trend implied that while Fargate's containerized architecture provided more consistent and efficient processing for bigger payloads, probably because of superior I/O management and resource allocation, Lambda's lightweight function invocation paradigm delivered quicker reaction times for smaller inputs.

3.3. Throughput and Scalability

The platforms were evaluated for their ability to handle concurrent invocations and maintain performance under load.

Table 3: Max Throughput (Requests/sec)

Concurrent Users	AWS Lambda	AWS Fargate
50	48 req/s	45 req/s
100	92 req/s	87 req/s
200	178 req/s	165 req/s

Figure 3: Max Throughput (Requests/sec)

The throughput findings under different concurrent user loads showed that AWS Lambda regularly processed more requests per second than AWS Fargate. Lambda handled 48 requests per second at 50 concurrent users, just exceeding Fargate's 45 requests per second. With rising concurrency, this performance difference grew; at 100 and 200 users, respectively, Lambda reached 92 and 178 requests per second, whereas Fargate managed 87 and 165 requests per second for the same loads. These results suggested that while Fargate's task-based scaling was more resource-intensive and somewhat slower, Lambda's architecture let it scale more aggressively and quickly by duplicating function instances to satisfy demand. Therefore, Lambda seemed more appropriate for tasks needing fast elasticity and great concurrency.

3.4. Cost Analysis

Cost-effectiveness was evaluated based on 1,000 inference executions per configuration.

Table 4: Cost per 1,000 Requests (USD)

Configuration	AWS Lambda	AWS Fargate
512MB / 0.25 vCPU	0.36	0.44
1024MB / 0.5 vCPU	0.49	0.48
2048MB / 1 vCPU	0.72	0.65

Table 4 shows the price per thousand requests for AWS Fargate and AWS Lambda under various resource settings. Charging $0.36 to Fargate's $0.44, AWS Lambda was more cost-effective at the lowest configuration (512MB / 0.25 vCPU). But, as the resource allocation rose to 1024MB / 0.5 vCPU, the prices for both systems became almost the same, with Lambda costing $0.49 and Fargate marginally lower at $0.48. Fargate, at $0.65 per 1,000 requests, was more affordable at the top setup (2048MB / 1 vCPU) than Lambda's $0.72. These results indicated that while Lambda's pay-per-invocation pricing model offered better cost-efficiency for smaller and short-duration tasks, Fargate's pricing structure became more advantageous for workloads requiring higher compute resources and longer execution times. This underlined the need of matching platform selection with certain workload traits to maximize operational expenses.

CONCLUSION

For event-driven machine learning inference tasks, this work offered a thorough performance benchmarking of AWS Lambda and AWS Fargate. The findings showed that depending on the workload features and deployment goals, each platform had unique benefits. For latency-sensitive apps needing fast container startup, AWS Fargate was a more appropriate option since it regularly beat AWS Lambda in cold start times. Conversely, AWS Lambda demonstrated better execution latency for smaller payloads and provided greater dynamic scalability under high concurrency, stressing its effectiveness for lightweight, burst-driven ML jobs. Throughput-wise, both systems scaled well, but Lambda's stateless, function-based design caused it to have a more aggressive auto-scaling reaction. Cost studies showed that although AWS Lambda was more affordable for low-resource, short-lived workloads, AWS Fargate grew more cost-effective for longer-running jobs at greater compute

allocations. The analysis found that individual workload needs—especially with regard to payload size, latency tolerance, concurrency levels, and cost sensitivity—should drive choice between AWS Lambda and Fargate. Using the best features of both systems, a hybrid deployment approach might potentially be considered to maximize performance and cost in practical ML applications.

REFERENCES

[1] A. Rose, Performance Evaluation of Serverless Object, Ph.D. dissertation, California State University, Northridge, 2023.

[2] B. N. Y. Arafath, A comparative study between microservices and serverless in the cloud, Master's thesis, OsloMet-storbyuniversitetet, 2022.

[3] C. Lekkala, "Containerization vs. Serverless Architectures for Data Pipelines," Serverless Architectures for Data Pipelines, Feb. 1, 2023.

[4] D. M. Naranjo, S. Risco, G. Moltó, and I. Blanquer, "A serverless gateway for event-driven machine learning inference in multiple clouds," Concurrency and Computation: Practice and Experience, vol. 35, no. 18, p. e6728, 2023.

[5] G. Kambala, "Cloud-Native Architectures: A Comparative Analysis of Kubernetes and Serverless Computing," 2023.

[6] I. Goswami, "Serverless Architecture for Machine Learning," 2023.

[7] J. J. Paul, Distributed Serverless Architectures on AWS, Berkeley, CA, 2023.

[8] L. Scotton, Engineering framework for scalable machine learning operations, 2021.

[9] M. Rahman, "Serverless cloud computing: a comparative analysis of performance, cost, and developer experiences in container-level services," 2023.

[10] M. Sisák, Cost-optimal AWS Deployment Configuration for Containerized Event-driven Systems, Ph.D. dissertation, 2021.

[11] N. Kodakandla, "Serverless Architectures: A Comparative Study of Performance, Scalability, and Cost in Cloud-native Applications," Iconic Research and Engineering Journals, vol. 5, no. 2, pp. 136-150, 2021.

[12] P. Grzesik, D. R. Augustyn, Ł. Wyciślik, and D. Mrozek, "Serverless computing in omics data analysis and integration," Briefings in Bioinformatics, vol. 23, no. 1, p. bbab349, 2022.

[13] S. Eismann, Performance Engineering of Serverless Applications and Platforms, Ph.D. dissertation, Universität Würzburg, 2023.

[14] S. R. Gallardo, Serverless strategies and tools in the cloud computing continuum, Ph.D. dissertation, Universitat Politècnica de València, 2023.

[15] V. Naik, Machine Learning Using Serverless Computing, 2021.