Concurrent Scaling: Evaluating AWS Lambda Performance through Load Testing

Balasubrahmanya Balakrishna[1]
bbsbems@gmail.com
ORCID: 0009-0000-1195-123X
Richmond, VA, USA

Abstract – In the dynamic environment of serverless computing, efficient concurrency management and reasonable utilization of load testing techniques closely correlate with performance improvement. This paper, aiming to offer practitioners practical insights and contribute to the theory of serverless architectures, intends to illuminate the intricate connection between concurrency settings and load testing results within AWS Lambda.

The paper thoroughly explores load testing and concurrency control in AWS Lambda. We specified a range of concurrency settings, including provisioned and unreserved, and meticulously selected representative Lambda functions. We subjected these functions to diverse simulated workloads using industry-standard load testing methods. We methodically measured and recorded performance metrics, including response times, throughput, and resource consumption, to identify significant patterns and correlations.

The article shows discrete performance quirks associated with various concurrency settings by exposing necessary trade-offs and optimization techniques. In addition to providing helpful advice for developers and architects, this research advances our theoretical knowledge of how concurrency affects the performance of serverless functions. Focusing on the unique features of AWS Lambda, the need for customized load-testing approaches is emphasized, offering practitioners practical advice on how to improve the efficiency and scalability of their serverless apps.

This article uniquely enriches the confluence of theory, practice, and policy in serverless computing. Defining the complex relationship between concurrency and load testing and providing a platform for future research enhances theoretical underpinnings. The results provide developers and architects with explicit guidelines on configuring concurrency settings to meet workload needs. Additionally, the study supports a sophisticated approach to load testing and concurrency control in the larger context of cloud computing, which adds to policy discussions surrounding serverless best practices.

Key Words– Lambda Concurrency, Load Testing AWS Lambda, Artillery

INTRODUCTION

Several studies indicate that, in terms of web application performance, users perceive response times in specific ways: they consider response times of less than 100ms as instantaneous, view delays ranging from 100ms to 300ms as predictable, experience a loss of attention at 1s, anticipate a response by 2s, and are inclined to abandon the application if the response time extends to 3s.[1]

When user expectations for web application performance are matched with the world of serverless computing, as demonstrated by AWS Lambda, the importance of load testing Lambda functions is apparent. Response times are crucial in determining the user experience, as perceptions influence. Load testing is a must to ensure that Lambda functions, which are crucial to online applications, can reliably meet or surpass the performance standards set by users. It is critical to use load testing to comprehend the complex link between concurrency, scalability, and response times in Lambda functions.

LAMBDA CONCURRENCY AND LOAD TESTING

In AWS Lambda, concurrency refers to the number of simultaneous function executions. The available concurrency for a Lambda function is the number of requests that can be processed concurrently by that function. This concept is closely related to load testing, where you simulate various levels of traffic to assess the performance and scalability of your application or service. Let's explore how each concept is related:

A. Lambda Concurrency

a. *Provisioned concurrency*: can be set for a Lambda function to guarantee a consistent availability of a defined number of instances for request handling. Provisioned concurrency proves beneficial when the aim is to reduce cold starts and maintain performance at the anticipated level.
b. *Unreserved Concurrency*: The remaining concurrency not allocated to provisioned concurrency is considered unreserved. This concurrency is subject to automatic scaling based on demand.

B. *Load Testing*
a. *Simulation of Traffic*: Load testing involves simulating various levels of user traffic to your application or service to observe its behavior under different conditions. This can include varying the number of simultaneous requests, the requests' rate, and the requests' complexity.

C. *Relation Between Concurrency and Load Testing*
a. *Scalability Testing:* Load testing Lambda functions aids in comprehending their scalability as traffic levels rise. Testing the behavior of functions under both average and peak loads is recommended. This testing also aids in determining the throttle behavior when the traffic increases by *x%* for over *y duration* during the peak traffic time.
b. *Cold Start Performance:* Concurrency settings, particularly for provisioned concurrency, may impact Lambda function performance during cold starts. Load testing assists in determining the speed at which functions scale to manage heightened loads and assesses the effectiveness of provisioned concurrency in reducing cold start latency.

D. *Optimizing Concurrency for Performance*
a. *Concurrency Adjustments:* The load testing results may indicate whether your Lambda function uses the available concurrency efficiently. Based on these results, you should adjust concurrency settings to optimize performance.
b. *Scaling Behavior:* Load testing unveils the scaling behavior of your Lambda functions, providing insights into the service's responsiveness to changes in demand. It enables an understanding of the speed at which the service can adapt and assesses whether the concurrency settings align with your application's requirements.

E. *Monitoring and Observability*
a. *Performance Metrics:* Monitoring and observability tools often accompany load testing, offering insights into the performance metrics of your Lambda functions. This valuable information aids in making informed decisions about concurrency settings.

LOAD TESTING IN SERVERLESS ENVIRONMENTS

This section introduces the practice of load testing for serverless applications, highlighting Artillery[2] as a load-testing framework among various options. Employing Artillery to load test a Lambda RESTful API on AWS. The goal involves configuring Artillery to simulate user interactions, capture response times, and other metrics[3].

A. *Case Study: Load Testing a Lambda RESTful API*

Let us consider a simple Lambda RESTful API hosted in AWS, which, on invoke process, produces a response as shown in Figure 1:

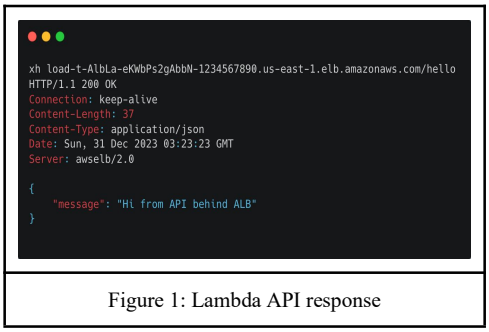

Figure 1: Lambda API response

Using a simple Artillery configuration, as shown in Figure 2, invoking Lambda function at a rate of 5 per second, ramping load up to 3 per second.

```
config:
  target: 'http://load-t-AlbLa-eKWbPs2gANbN-579184009.us-east-1.elb.amazonaws.com'
  phases:
    - duration: 120
      arrivalRate: 5
      rampTo: 3
  processor: "./random.js"
  defaults:
    headers:
      Client-Correlation-id: {{ ccid }}
scenarios:
  - flow:
    - function: "generateRandomData"
    - get:
        url: "/hello"
    - log: "Sent GET request to /hello with {{ ccid }}"
```

Figure 2: Artillery configuration

Artillery will run the test, launching new virtual users as defined under config, phases block as shown in Figure 2.

The test generates several reports, but a few important ones for the discussion are shown in Figures 3 and 4 below. Figure 3 shows that the test invoked the Lambda function 480 times with 0 errors. In this scenario, the load was ramped from 1 TPS to 5 TPS to simulate traffic ramp-up.

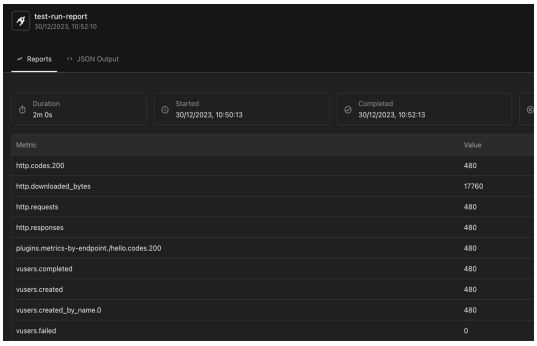

Figure 3: Artillery Test Report

Artillery shows the median response time was 147 ms, p95 was 468.8 ms, and p99 was 772.9 ms.

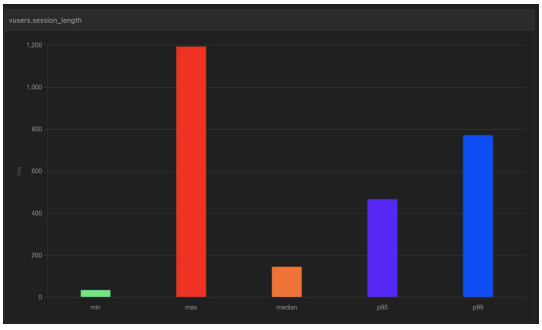

Figure 3: Artillery Test Metric Report

This result gives insight into Lambda metrics under load. The recommended approach is to look at some of the metrics like Invocations, Concurrent Execution, and Duration metrics from the AWS console, which for this run looks as shown in Figure 4:

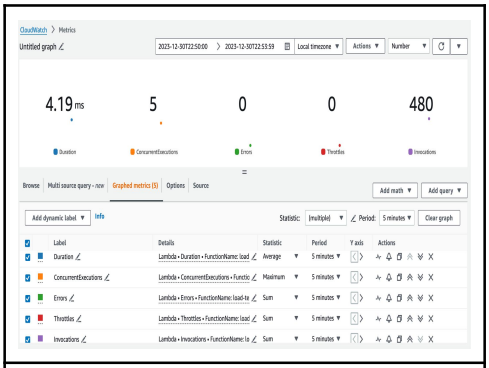

Figure 5: AWS Cloudwatch Metrics.

Let us run a slightly bigger load with a TPS of 15 by updating the rampTo value to 15. The test produces below Artillery and AWS metrics:

Artillery reports, in Figure 6 and Figure 7, show 1200 successful tests against 480 tests in the previous run. The median response time for this run is 130 ms, p95 is 450 ms, and p99 is 528.6 ms.

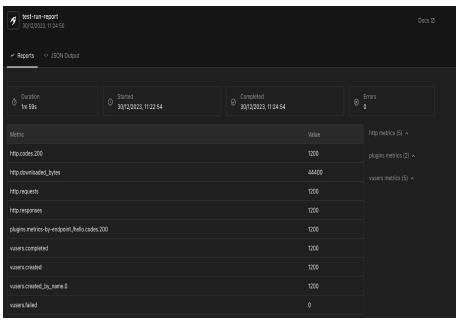

Figure 6: Artillery Test Report

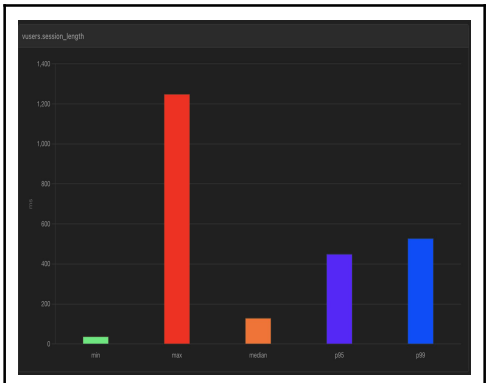

Figure 7: Artillery Test Metric Report

Looking at the AWS metrics for the second run, shown in Fig 8

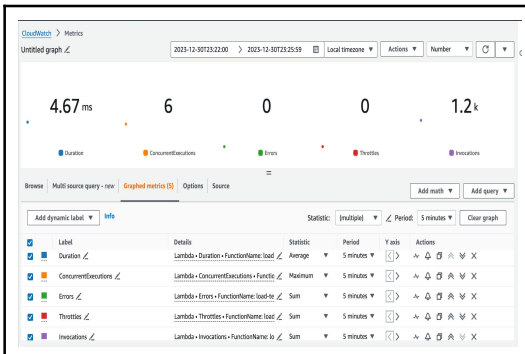

Figure 8: AWS Cloudwatch Metrics,

It is observable that the function's duration extended in response to heightened load. In the initial minute, the function executed for approximately 5 seconds, gradually reaching a peak of 15 seconds before stabilizing for the remainder of the load test.

More TPS sometimes equals to more concurrency and Concurrency is a point-in-time measurement , and not the same as request per second.

Long-running functions require more concurrency.

Hence, concurrency is calculated as:

Concurrency = TPS x Duration (in seconds).[4]

For example,

10 RPS x 500 ms = 5 concurrency, and

10 RPS x 1s = 10 concurrency.

Using CloudWatch performance metrics and a load-testing framework, we can determine the optimal concurrency for your Lambda function. This involves generating production-level or higher loads to assess and define the desired (reserved) concurrency for the specific function under consideration.

It's important to highlight that load-testing frameworks such as Artillery drive synchronous invocations, impacting Lambda function duration, TPS, and concurrency. This dynamic would differ if we were to use asynchronous invocations with Lambda.

By executing the function asynchronously (using headers X-Amz-Invocation-Type: Event and X-Amz-Log-Type: None) and rerunning the load test, we promptly achieved 3 concurrency, ramping up from that point. In contrast, synchronous invocation commenced with approximately 1 concurrency. Therefore, an asynchronous invocation is favored, enabling quicker bursts and benefiting from AWS's built-in retry logic.

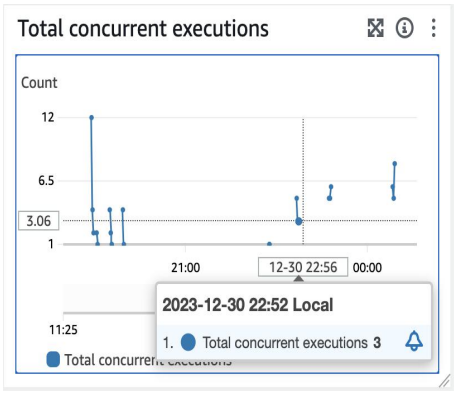

Figure 9: Aysnc invocation of Lambda function

CONCLUSION

In conclusion, exploring load testing in serverless environments, mainly focusing on AWS Lambda functions, underscores the critical role of meticulous performance evaluation in optimizing application responsiveness. Leveraging frameworks like Artillery, developers can navigate the complexities of concurrency, scalability, and user-centric response times. The case study involving the load testing of a Lambda RESTful API on AWS provides tangible insights into Artillery's effectiveness in assessing a function's ability to handle production-level loads.

The nuanced understanding gained through this exploration is crucial for making informed decisions on concurrency settings and ensuring that Lambda functions align with user expectations for instantaneous, predictable, and timely responses.

The observation that asynchronous invocation yields superior concurrency performance compared to synchronous invocation further emphasizes the practical implications of load-testing choices.

This exploration of load-testing techniques, metrics analysis, and performance optimization gives developers useful knowledge about serverless architectures and emphasizes the importance of using the appropriate instruments for precise evaluations. Load testing becomes an essential technique as serverless computing develops, helping to produce scalable and dependable services that satisfy users in ever-changing and dynamic contexts.

REFERENCES

[1] (n.d.). *Defining the Core Web Vitals metrics thresholds*. Web.dev. https://web.dev/articles/defining-core-web-vitals-thresholds?continue=https%3A%2F%2Fdevelopers.google.com%2Flearn%2Fpathways%2Fweb-vitals%23article-https%3A%2F%2Fweb.dev%2Fdefining-core-web-vitals-thresholds

[2] Artillery Software, Inc. (n.d.). *Artillery Docs*. Https://www.Artillery.io/Docs. https://www.artillery.io/docs/get-started/first-test

[3] AWS (n.d.). *Important metrics for CloudWatch*. AWS Lambda Operator Guide. https://docs.aws.amazon.com/lambda/latest/operatorguide/important-metrics.html

[4] AWS (n.d.). *Lambda function scaling*. AWS Lambda Developer Guide. https://docs.aws.amazon.com/lambda/latest/dg/lambda-concurrency.html

AUTHOR INFORMATION

Balasubrahmanya Balakrishna is a Senior Lead Software Engineer, serverless-focused strategy leader, open-source enthusiast, and cloud-native advocate.

FUNDING STATEMENT

This paper received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.