

Evaluating the performance resilience of serverless applications using chaos engineering

Ahmad Zayed¹ and Amro Al-Said Ahmad^{2,*}

¹ Faculty of Information Technology, Philadelphia University, Amman, Jordan

² School of Computer Science and Mathematics, Keele University, Staffordshire, United Kingdom

ahmadomar19984@gmail.com, a.m.al-said.ahmad@keele.ac.uk

*corresponding author

Abstract—This study explores the use of chaos engineering in evaluating the performance and resilience of serverless applications, which are built as complex distributed systems subject to different types of failures and errors. By intentionally injecting controlled failures and uncertainty into the system, we evaluate the impact of random delay injections on Lambda functions using two methods: one during code execution and the other during runtime. Our research aims to evaluate the performance and resilience of Lambda functions under controlled chaos experiments and study the impact of faults on serverless applications' behaviour by comparing the results with the baseline data regarding performance metrics.

Keywords- Resilience; Performance; Delay injection; Serverless Functions; Chaos Engineering.

1. INTRODUCTION

Cloud computing has rapidly grown, enabling companies to build modern distributed systems with high performance and scalability. Cloud-native Serverless architecture allows developers to deploy applications without managing servers, reducing costs, and increasing performance and reliability. The increasing use of complex systems and cloud solutions, such as serverless solutions, requires thoroughly examining their dependability and resilience. Cloud services are prone to failures that can affect other resources, making integrating dependability and resilience in the development process crucial. Fault injection, a part of software testing, is used to assess the dependability of software systems and is frequently utilized to evaluate cloud system reliability. Several major technology companies have adopted Chaos Engineering, which is a method that intentionally introduces controlled failures into a system to test its resilience [1]. This approach is crucial for evaluating the performance of serverless applications built on cloud infrastructure, as they can be vulnerable to various types of failures and errors. By testing applications under different fault scenarios, developers can identify and fix potential vulnerabilities, enhance fault tolerance, and ensure correct functionality under unexpected events. Chaos testing provides insight into application behaviour under real-world fault scenarios, supporting performance and fault tolerance. Our research investigates how controlled failures impact the performance resilience of serverless systems. We measure performance metrics and compare the results of applications with and without chaos

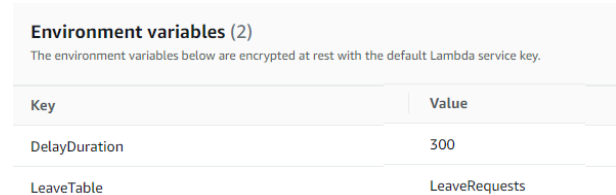
engineering. The evaluation shows that chaos engineering affected performance, resulting in decreased quality of cloud services, such as average response time.

2. METHODOLOGY AND EXPERIMENTAL SETTING

Chaos engineering is a technique used to test the resiliency of systems by intentionally introducing controlled failures or disruptions, such as delays, to see how the system responds and identify any potential issues. The two techniques used in this paper are:

Code Level Chaos Engineering:

The introduction of a delay in Lambda functions is a form of chaos engineering during code execution [2]. This type of injection can have positive and negative effects on performance resilience. On the one hand, it helps identify potential issues or bottlenecks. Still, on the other hand, it can lead to performance issues by slowing down the overall execution time of the application. The delay code will be added to the application's codebase and deployed to production to run every time the function is invoked. The environment variable used in AWS to introduce a delay (in milliseconds) into Lambda functions, as shown in Figure 1.



Environment variables (2)	
The environment variables below are encrypted at rest with the default Lambda service key.	
Key	Value
DelayDuration	300

Figure 1. Delay Environment Variable

Chaos Engineering at the application level (runtime):

The fault injection methodology involves injecting slower connections into the system under test [3]. We use Charles version 4.6.5 to simulate slower connection latency in milliseconds (ms). This helps analyse how applications respond to slower dependencies, potentially causing request accumulation and service interruptions. We set random delay latencies in ms using Charles (www.charlesproxy.com/) for each request, which is 300 ms in this paper.

Experiments Design and Testing Process:

We tested two serverless functions with and without chaos engineering using JMeter, with four different numbers of users: 150, 300, 600, and 1200. Our metrics included response time, latency, throughput, and lambda duration (min, average and max). We injected 300 ms delays using two techniques

and compared the impact on performance and resilience with over 240 experiments conducted.

We used LeaveWebApp¹, an Open-source Serverless application hosted on AWS to demonstrate our methodology. The application is built on native AWS services with decoupled architecture design principles and was published on GitHub by a Solutions Architect at AWS, with two serverless functions: Add Leave Request (POST) and Get Leave Request (GET). The application was hosted using AWS-native services on Elastic Container with 8vCPU, 24GB, and DynamoDBv2. The API Gateway Throttling was 10000 requests per second with a burst of 5000 requests.

3. EXPERIMENTAL RESULTS AND ANALYSIS

The paper compares the performance of serverless functions in a normal state and with chaos injection. Our results show that the performance of the POST and GET functions has dropped in terms of response times, throughput, and latency values in both fault injection experiments. Further, we note similar behaviour regarding Lambda functions' duration times. Our experiments show that the overall Throughput dropped between 15% and 17% compared to the baseline, overall response times and latency dropped by 66%, and the Lambda functions' execution times increased between 3% and 14%. Figure 2 shows the behaviour of both functions to the injected faults in terms of function execution times. Although we can see similar behaviour in the Add Leave function, values were still different, especially regarding the average and maximum Lambda durations. This demonstrated that the same function acted differently in response to the injected faults despite the same value.

The Get Leave function (GET) behaved differently for both types of injections. Figure 2b shows that in some cases, the Lambda durations recorded higher values for the slow connections' experiments, although the same 300ms injections were used in both functions. In the experiments with slow connection injection for the Add Leave function, the

response time and latency values were higher than the chaos injection within the function itself (i.e., 300 ms injection at code level). Figure 2 also highlights an important aspect of application behaviour, which can be influenced by the type of fault or chaos that was injected within the system. This observation is particularly interesting because it reveals that different types of HTTP APIs or Serverless functions can react differently to the same fault even when subjected to identical experimental settings. This underscores the importance of considering the unique characteristics of each application when assessing the potential impact of faults and chaos on its behaviour. By doing so, developers and operators can better anticipate and address issues that may arise, leading to a more stable and reliable system overall.

4. CONCLUSIONS

The study findings indicate that the impact of the code-level injection technique and the slow connection injection with 300 ms random delay on both functions is similar in terms of response time, latency, and throughput. However, the 300 ms injection within the functions (code level) had a greater impact on the duration of the Lambda than the effect of a slow connection. This highlights how different faults can affect the application's performance and resilience in various ways. The results of our study provide insights into the performance resilience of applications and the effectiveness of chaos engineering in evaluating serverless solutions.

REFERENCES

- [1] A. Basiri *et al.*, "Chaos Engineering," *IEEE Softw.*, vol. 33, no. 3, pp. 35–41, 2016, doi: 10.1109/MS.2016.60.
- [2] A. Al-Said Ahmad, L. F. Al-Qora'n, and A. Zayed, "Exploring the impact of chaos engineering with various user loads on cloud native applications: an exploratory empirical study," *Computing*, pp. 1–37, 2024, doi: 10.1007/s00607-024-01292-z.
- [3] A. Al-Said Ahmad and P. Andras, "Scalability resilience framework using application-level fault injection for cloud-based software services," *J. Cloud Comput.*, vol. 11, no. 1, pp. 1–13, 2022, doi: 10.1186/s13677-021-00277-z.

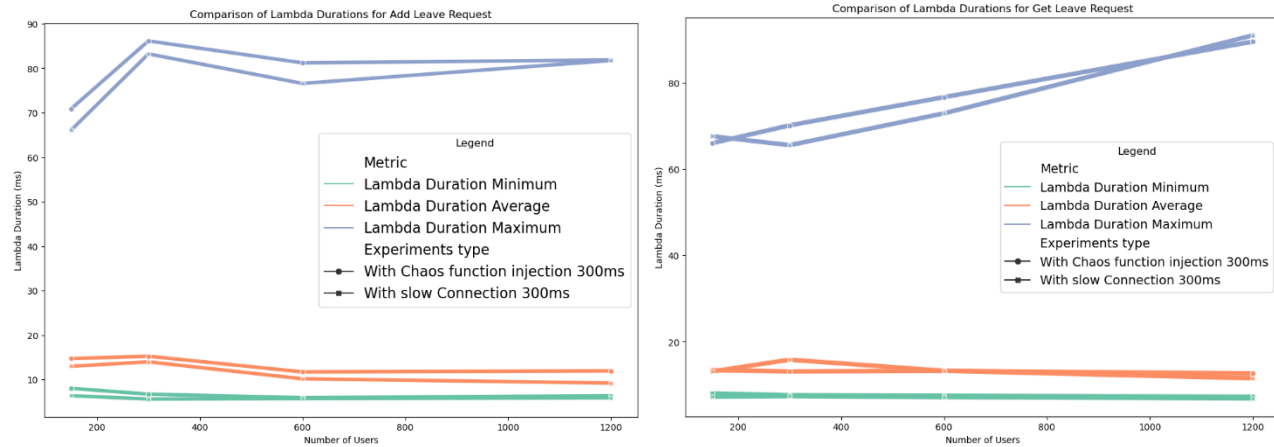


Figure 2. Comparisons between faults impact on Lambda functions: a) Add Leave Request and b) Get Leave Request

¹ <https://github.com/aws-samples/aws-serverless-workshop-decoupled-architecture>