*International Journal on Recent and Innovation Trends in Computing and Communication*
*ISSN: 2321-8169 Volume: 11 Issue: 9*
***Article Received:*** *25 July 2023* ***Revised:*** *12 September 2023* ***Accepted:*** *30 September 2023*

# Execution of Serverless Functions Lambda in AWS Serverless Environment

**Shashank Srivastava[1*], Bineet Kumar Gupta[1],Dheeraj Tandon[2], Ankit Kumar Singh[2],Mohammad Husain[3], Arshad Ali[3], Sami Alshmrany[3], Sanjay Gupta[4], Sandeep Dubey[4]**

[1*,1]DCSIS,IOT, Shri Ramswaroop Memorial University, Lucknow- Deva Road Barabanki,Uttar Pradesh,India
[2]School of Engineering, Department of Computer Science & Engineering, Babu Banarsi Das University, Lucknow (UP), India
[2]DCSIS,IOT, Shri Ramswaroop Memorial University,Deva Road , Barabanki,Uttar Pradesh,India
[3]Islamic University of Madinah ,Al Madinah ,Al Munawarah, Saudi Arabia
[4]Vice President HCL Technologies
[4]Director - OD SOFTECH Ltd , UK
[1*]shivam.shashank@gmail.com, [1]hod.dcsis@srmu.ac.in, [2]dheerajtandon9@gmail.com,
[2]ankitkumar.ids@srmu.ac.in,[3]mohd.husain90@gmail.com,[3]a.ali@iu.edu.sa,[3]s.alshmrany@iu.edu.sa,[4]sanjay_gupta@hcl.com,
[4]Sandeepdubey@icloud.com

**Abstract—** Serverless computing, epitomized by AWS Lambda, has revolutionized application development and deployment by abstracting away server management and offering auto-scalability. AWS Lambda's 15-minute maximum timeout for function execution, however, presents a unique challenge for users aiming to process longer tasks within the serverless framework. In this paper, we dive deep into the execution dynamics of AWS Lambda within the AWS serverless environment, with a particular focus on extended execution times. As serverless computing gains traction understanding the nuances and limitations of AWS Lambda's default settings becomes paramount.

This paper investigates different request handling mechanisms within AWS Lambda and presents empirical data to demystify the underlying processes. By addressing the challenges posed by long-running serverless functions, we aim to provide practical insights and potential solutions for developers and architects seeking to optimize their serverless applications. In the quest for efficient triggering mechanisms for extended serverless functions, this paper offers valuable guidance, empowering users to leverage AWS Lambda effectively while circumventing the default timeout constraints.

**Keywords-** Serverless Computing, Serverless Function, AWS, AWS Lambda, Cloud Computing.

## I. INTRODUCTION

Serverless computing has emerged as a transformative paradigm in the world of cloud computing, redefining how applications are developed, deployed, and managed. At the forefront of this revolution is Amazon Web Services (AWS) Lambda, a serverless compute service that has rapidly gained traction among developers and enterprises alike. AWS Lambda offers the promise of effortlessly executing code without the need to provision or manage servers, making it an attractive choice for building highly scalable and cost-effective applications. Serverless computing has gained widespread popularity for its scalability, cost efficiency, and developer-friendly approach. However, efficient execution and management of serverless functions heavily depend on effective triggering mechanisms. AWS Lambda, a prominent serverless platform, enforces a maximum timeout limit of 15 minutes for function execution, posing challenges for handling longer requests. When users need to execute code that exceeds this limit, they encounter significant obstacles.

To dive deep into the execution dynamics of AWS Lambda within the AWS serverless environment, with a particular focus on extended execution times is very important part as serverless computing gains traction, understanding the nuances and limitations of AWS Lambda's default settings becomes paramount.

There are different request handling mechanisms within AWS Lambda and presents empirical data to demystify the underlying processes. By addressing the challenges posed by long-running serverless functions, we aim to provide practical insights and potential solutions for developers and architects seeking to optimize their serverless applications.

### A. Key Characteristics of Serverless Computing

**Event-Driven Architecture**: Serverless applications are built around triggers or events, and functions are called in response to things like HTTP requests, database changes, file transfers, or scheduled tasks. Event-driven design promotes excellent decoupling and smooth scalability of this system

**Automatic Scaling**: Automatic scaling on serverless platforms based on incoming workloads. Adaptive scaling of functions up or down according to demand ensures the best possible use

*International Journal on Recent and Innovation Trends in Computing and Communication*
*ISSN: 2321-8169 Volume: 11 Issue: 9*
***Article Received:*** *25 July 2023* ***Revised:*** *12 September 2023* ***Accepted:*** *30 September 2023*

of resources and continuous availability. This scalability is achieved without manual configuration or capacity planning.

**Usage-based billing**: In serverless computing, users are billed only for the resources they actually use. The number of function calls and the length of each call are the most important factors that determine billing. Users pay only for the exact time their features are running under this pricing method, saving money on wasted resources.

**Stateless Execution**: Serverless functions run stateless, without maintaining persistent state between calls. This statelessness enables horizontal scaling and ensures that each call is autonomous and independent.

**Event-driven scaling:** Event-driven scaling: serverless solutions adapt resource allocation to changing workload in real time. The platform automatically allocates more computing resources as the number of transactions increases to meet the increased workload. This event-driven scaling ensures that resources are used efficiently and that fluctuating workloads can be handled without manual intervention.

**Better Developer Productivity**: Serverless computing centralizes the complexity of infrastructure management, allowing developers to focus only on building application logic. This eliminates operational costs associated with server administration, management and scaling, which speeds up development cycles and increases developer productivity.

**Strong fault tolerance and high availability:** Serverless platforms handle fault and high availability perfectly. Functions are automatically replicated across multiple Availability Zones, providing flexibility and fault tolerance. In the event of resource failure or unavailability, the platform seamlessly redirects transactions to available resources and maintains uninterrupted operations.

**Developer Productivity:** Serverless computing abstracts away the infrastructure management, allowing developers to focus solely on writing application logic. It reduces the operational burden by eliminating the need to provision, manage, and scale servers, enabling faster development cycles and improved developer productivity.

**Fault Tolerance and High Availability**: Serverless platforms handle fault tolerance and high availability transparently. They automatically replicate functions across multiple availability zones, ensuring resilience and fault tolerance. In case of a failure or resource unavailability, the platform seamlessly redirects the events to available resources.

## II. RELATED WORK

In the realm of triggering serverless functions within serverless environments, various authors have contributed innovative approaches to address this challenge. One notable milestone in this field is the work titled "Computing without Servers" [3], which serves as a foundational piece by introducing the concept of sequencing serverless functions and delving into the intricacies of serverless function composition.

Expanding on this foundation, Rabbit made significant strides in 2017 by introducing serverless state machines under the banner of Conductor. In conjunction with this development, they also introduced a DSL called Composer, with the primary objective of simplifying the creation of state machines. These advancements aimed to streamline the orchestration of serverless functions within complex workflows.

Furthermore, Trapeze [16] introduced dynamic IFC (Information Flow Control) to the realm of serverless computing. This innovation was coupled with the implementation of sandboxes for serverless functions, which greatly enhanced their ability to interact with shared storage securely and efficiently.

In the pursuit of optimizing serverless computing, several initiatives [4] have focused on achieving elastic parallelization. Additionally, cloud orchestration frameworks [5] have introduced a specialized language tailored for managing cloud environments. Engage, as proposed by Fischer et al. in 2012, functions as a deployment manager specifically designed to support inter-machine dependencies, contributing to the efficiency of serverless deployments.

Moreover, [6] introduced an embedded DSL (Domain-Specific Language) aimed at simplifying the development of cloud-configurable programs. Meanwhile, CPL [7] presents a unified language that facilitates the creation of distributed cloud programs and the implementation of their distribution routines, enhancing the overall scalability of serverless applications.

In contrast to these approaches, Jangda et al. [14] presented serverless computing semantics that leverage a key-value store for persistence, employ transactions to handle concurrency, and utilize unique identifiers to ensure secure reinvocation. These strategies, grounded in the principles of the Naïve Bayes theorem, provide a robust foundation for addressing critical challenges in serverless computing.

As part of our research endeavours, we aim to build upon these insights and pioneer an improved mechanism for more effectively addressing the challenges associated with

*International Journal on Recent and Innovation Trends in Computing and Communication*
*ISSN: 2321-8169 Volume: 11 Issue: 9*
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

triggering serverless functions within the serverless environment.

## III. PROBLEM DEFINITION:

I Serverless systems, any container can exist in two states. The container will be in one of two states: idle or active. The idle state of a container shows that it is not currently performing anything, whereas the active state indicates that it is actively executing server less operations.

The first obstacle appears after a few minutes of inactivity; at this point, all global variable information is misidentified or lost. This problem arises when serverless services are run in a container that may keep a short-term state during execution and then abruptly shut down when the container reaches the idle state. Because developers are unaware of when a container shuts down by reaching its idle state, this information is not available to them.The Progress of serverless functions in Serverless Environment is as Follows:

### *A.* State of flux & Timeout Limits

In AWS serverless Environment the maximum timeout for a **serverless function execution is 15 Minutes**, previously it was 05 minutes [Source AWS Official Website]. Therefore when any user wants to execute long requests means the request which will take more than 15 minutes for its execution, it cannot be execute in one shot because after 15 minutes of execution, it will be automatically stopped & timeout error message will pop up.

We have scientifically analyzed this situation & found it correct that for the request having less than 15 minutes , we get the desire result & for the request having more than 15 minutes responded Time out error message.

## IV. EXPERIMENTAL FRAMEWORK

To perform the experiment on Server less Environment, we have chosen the following specifications, which have been shown in Table1.

| S. No. | Requirement | Specifications |
|---|---|---|
| **1** | Programming Language | Node.js |
| **2** | Serverless Environment | AWS |
| **3** | Serverless Function | AWS Lambda |

Table 1: Experimental Specification

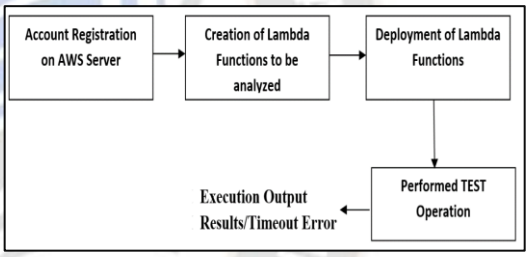
***Steps:***

To analyze & identify this situation, we have performed following Steps:

1. Created an online Account in AWS Server by performing Account Registration Formalities.
2. Created 3 Lambda Functions; Lambda_A, Lambda_B,Lambda_C. For creating these functions, we have chosen Node.js as programming language.
3. After creating these 3 functions, We have deployed these functions in AWS Server for analyzing & identifying their progress.
4. After deploying these 3 functions in AWS, we have performed Test operation for the results.
5. After getting Results, we have observed their Execution metrics on different parameters with time.

## V. PROCESS IDENTIFICATION FLOW:

In the Process Identification Flow, we have described the flow of execution of our experimental setup.




**Coding: Node.js**

**Creation of Lambda_A Function:**

During our experimental Setup, the main role of Lambda A function is to call/Request the functions to be executed in Serverless Environment.

**Creation of Lambda_B Function:**

The Lambda B function holds the execution, which is going to take more than 15 minutes.

**Creation of Lambda_C Function:**

The Lambda C function holds the execution, which is going to take less than 15 minutes.

***Algorithm of Lambda_A: [Calling Function]***

```
var AWS = require('aws-sdk');
//AWS.config.region = 'eu-west-1';
var lambda = new AWS.Lambda();
exports.handler = function(event, context) {
```

*International Journal on Recent and Innovation Trends in Computing and Communication*
*ISSN: 2321-8169 Volume: 11 Issue: 9*
***Article Received:*** *25 July 2023* ***Revised:*** *12 September 2023* ***Accepted:*** *30 September 2023*

```
 await new Promise(resolve =>setTimeout(resolve,
20000));
  var params = {
FunctionName: 'Lambda_B', // Change to
Lambda_B will give timeout error
InvocationType: 'RequestResponse',
LogType: 'Tail',
   Payload: '{ "name" : "SRMU" }'
  };
lambda.invoke(params, function(err, data) {
   if (err) {
context.fail(err);
   } else {
context.succeed(params.FunctionName+ ' said '+
data.Payload);
   }
  });
};
```

***Algorithm of Lambda_B:***

For Testing of Function of Longer Requests **[More Than 15 Minutes]**

```
exports.handler = async function(event, context) {
  await new Promise(resolve =>setTimeout(resolve,
899900));
console.log('Lambda B Received event:',
JSON.stringify(event, null, 2));
context.succeed('Hello ' + event.name);
};
function sleep(ms) {
  return new Promise((resolve) => {
setTimeout(resolve, ms);
  });
}
```


Figure 1: AWS Response Message for longer request processing

In the figure 1, we have shown the response generated by the lambda A function. It omits the **Failed** status while executing the longer request i.e the request, which takes more than 15 minutes of processing.

***Algorithm of Lambda_C:***

For Testing of Function of Small Requests **[Less Than 15 Minutes]**

```
exports.handler = async function(event, context) {
  await new Promise(resolve =>setTimeout(resolve,
7000));
console.log('Lambda C Received event:',
JSON.stringify(event, null, 2));
context.succeed('Hello ' + event.name);
};
function sleep(ms) {
  return new Promise((resolve) => {
setTimeout(resolve, ms);
  });
}
```

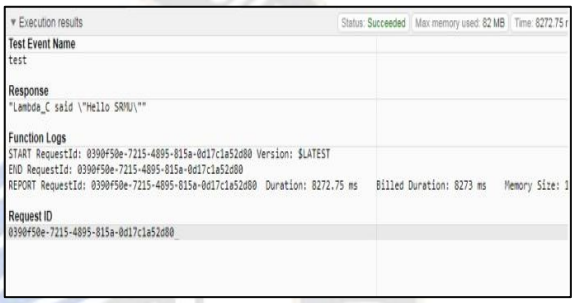


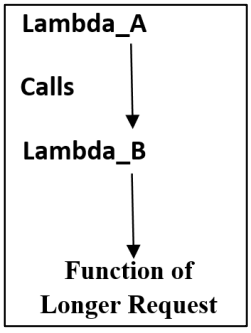

Figure 2: AWS Response Message for Small Size request processing

In the figure 2, we have shown the response generated by the lambda A function. It omits the **Succeeded** status while executing the request i.e the request, which takes less than 15 minutes of processing.

## VI. EXECUTION FLOW DIAGRAM

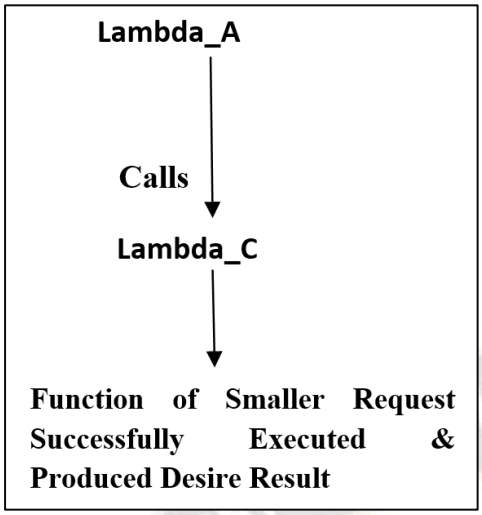
In the execution flow diagram, we have shown the process of our experimental setup to analyze the processing of different size requests in serverless container, which are been processed by serverless function.

***Execution of Lambda B Function by Function Lambda A***



*International Journal on Recent and Innovation Trends in Computing and Communication*
*ISSN: 2321-8169 Volume: 11 Issue: 9*
***Article Received:*** *25 July 2023* ***Revised:*** *12 September 2023* ***Accepted:*** *30 September 2023*

***Execution of Lambda C Function by Function Lambda A***




## VII. EXECUTION FLOW METRICS:

In the following figure 3, 4 we have shown the Execution Flow Metrics, In this have included the real time statistics on different parameters i.e Invocations & Duration & Throttles.

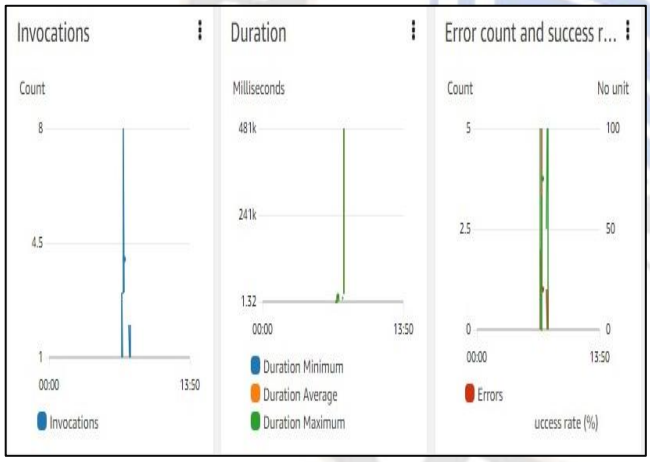



Figure 3: Invocation & Duration Metrics Graph


Figure 4: Throttles Metrics Graph

## VIII.CONCLUSION AND FUTURE SCOPE

In this paper we have analyzed the different situations for handling smaller & longer requests in AWS environment & identified that for longer requests which take more than 15 minutes of execution time, AWS server was unable to process & omits an error. Therefore, Effective methodology for Triggering of serverless function is very important in the serverless Environment for developers to use the serverless platforms in executing their program function due the distinct features of the serverless computing. In proposed paper provides the environment where triggering of serverless functions can be more compatible. With these platforms easily and can address the challenges related to serverless function Execution in serverless platform because demand and acceptability of Public Cloud platforms are growing very fast with time. The handling the longer requests of to process the different type of documents, it would be very necessary to develop the effective solutions to handle them. This can be more beneficial for both the developers and service providers.

**Acknowledgment:**
The researchers wish to extend their sincere gratitude to the Deanship of Scientific Research at the Islamic University of Madinah, Saudi Arabia for the support provided to the Post-Publishing Program 3.

We would like to express our sincere gratitude to our supervisor, Dr. Bineet Kumar Gupta, Associate Professor, DCSIS, IOT, SRMU Lucknow for his valuable guidance and support throughout the research process. His expertise and insights were invaluable in shaping our research and helping me to overcome challenges. We also want to thank our SRMU University, Lucknow for their helpful feedback and support

**References**

[1]. Panda, Surya, Mehta, Ashima. (2018). Design of Infrastructure as a Service (IAAS)Framework with Report Generation Mechanism, International Journal of Applied Engineering Research, 0973-4562 V.13, N. 2 (2018) pp. 942-946
[2]. Gurudatt Kulkarni, Prasad Khatawkar, Jayant Gambhir, Cloud Computing Platform as Service, International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-1, Issue-2, December 2011
[3]. Espadas, J., Concha, D., and Molina, A., "Application Development over Software-as-a-Service platforms,"In Proceedings of Software Engineering Advances(ICSEA 2008), IEEE,pp.97 - 104, October, 2008.
[4]. Theo Lynn, PierangeloRosati, Arnaud Lejeune, and Vincent C. Emeakaroha. A preliminaryreview of enterprise serverless cloud computing (function-as-a-service) platforms. In IEEEInternational Conference on Cloud Computing Technology and Science, CloudCom 2017, HongKong, December 11-14, 2017, pages 162–169, 2017.

[5]. Choudhary, Brijesh&Pophale, ChinmayGutte, Aditya, Dani, Ankit, Sonawani, Shilpa.(2020). Case Study:Use of AWS[5] Lambda for Building a Serverless Chat Application.10.1007/978-981-15-0790-8_24.
[6]. Moroney, Laurence. (2017). Cloud Functions for Firebase. 10.1007/978-1-4842-2943-9_8.
[7]. D. Chappell. Introducing the Azure services platform. White paper, Oct. 2008.
[8]. Kuntsevich, Aleksandr&Nasirifard, Pezhman& Jacobsen, Hans-arno. (2018). A DistributedAnalysis and Benchmarking Framework for Apache OpenWhisk Serverless Platform. 3-410.1145/3284014.3284016.
[9]. Sampé, Josep&Vernik, Gil & Sánchez-Artigas, Marc &López, Pedro. (2018). Serverless DataAnalytics in the IBM Cloud. 1-8. 10.1145/3284028.3284029.
[10]. Shafiei, Hossein&Khonsari, Ahmad &Mousavi, P. (2019). Serverless Computing: A Surveyof Opportunities, Challenges and Applications. 10.13140/RG.2.2.32882.25286.
[11]. Heydari, Atefeh&Tavakoli, Mohammadali&Riazi, Mohammad. (2014). An Overview ofPublic cloud[11] Security Issues. International Journal of Management Excellence. 3.10.17722/ijme.v3i2.166.
[12]. Yunxia, Jiang & Bowen, Zhao &Shuqi, Wang &Dongnan, Sun. (2014). Research ofEnterprise Private cloud[12] Computing Platform Based on OpenStack. International Journal ofGrid and Distributed Computing. 7. 171-180. 10.14257/ijgdc.2014.7.5.16.
[13]. Aryotejo, Guruh, Daniel YeriKristiyanto and Mufadhol. “Hybrid cloud: bridging of private and public cloud computing.” Journal of Physics: Conference Series 1025 (2018): n. pag.
[14]. Bridging of private and public cloud, computing. Journal of Physics: Conference Series.1025. 012091. 10.1088/1742-6596/1025/1/012091.
[15]. Jangda, Abhinav, Pinckney, Donald & Baxter, Samuel & Devore-McDonald, Breanna;Spitze, Joseph, Brun, Yuriy, Guha, Arjun. (2019). Formal Foundations of ServerlessComputing,ACM Journal
[16]. Baldini, I. et al. (2017). Serverless Computing: Current Trends and Open Problems. In: Chaudhary, S., Somani, G., Buyya, R. (eds) Research Advances in Cloud Computing. Springer, Singapore. https://doi.org/10.1007/978-981-10-5026-8_1
[17]. Dutta, Pranay, Dutta, Prashant. (2019). Comparative Study of Cloud Services Offered byAmazon, Microsoft and Google. International Journal of Trend in Scientific Research andDevelopment. Volume-3. 981-985. 10.31142/ijtsrd23170
[18]. Chandran, S., Verma, S.B.: Touchless palmprint verification using shock filter SIFT I-RANSAC and LPD IOSR. J. Comput. Eng. 17(3), 2278–8727 (2015)