ISSN: 2754-6659

Journal of Artificial Intelligence & Cloud Computing

SCIENTIFIC Research and Community

Review Article

Open Access

Optimizing AWS Lambda Performance: Unveiling the Relationship between Memory Allocation, CPU Power and Cost Efficiency

Balasubrahmanya Balakrishna

Senior Lead Software Engineer, Richmond, VA, USA

ABSTRACT

This technical paper examines, emphasizing AWS Lambda, the crucial relationship between memory use, cost-effectiveness and performance in server less applications. We highlight the significance of memory allocation, look into factors that affect consumption, and offer methods for locating and resolving memory-related problems.

An essential part of our research is the relationship between CPU power assignment and Lambda execution time. We illustrate the significant influence of CPU power on execution time and operating expenses using a Lambda function from the real world. We present a sample Lambda function and examine its performance with different memory setups.

The AWS Lambda Power Tuning framework results, intended to identify the most economical memory configuration automatically are compared against manual memory determination in the article. Empirical evidence evaluates the framework's success in maximizing performance and cost.

The report concludes by synthesizing significant findings and providing practitioners with practical suggestions. This study adds significant knowledge to server less computing by offering real-world examples, empirical support, and contrasts between automated and human memory tuning. By addressing the direct relation of memory allocation to cost-effectiveness and its implications on server less application performance, the work advances our understanding of the complex balance necessary for optimal resource management.

***Corresponding author**

Balasubrahmanya Balakrishna, Senior Lead Software Engineer, Richmond, VA, USA.

Received: October 04, 2022; **Accepted:** October 14, 2022; **Published:** October 22, 2022

Keywords: AWS Lambda, AWS Step Function, AWS Power Tunning, Chudnovsky Algorithm

Background

Practioners can concentrate on writing code instead of worrying about maintaining the underlying infrastructure. Practitioners typically look into two main strategies to optimize function performance assigning the right amount of memory to a function and optimizing the speed at which code executes.

Code Optimization for Speed

For the best possible function performance, efficient code execution is essential. Various strategies, such as code reworking and algorithmic enhancements, to reduce execution times and boost productivity.

The Role of Memory Allocation

While optimizing Lambda functions, one must choose the optimal memory allocation. This study explores the proportional relationship between configured memory and the CPU allocated to a function, emphasizing how memory settings directly affect function performance and overall cost [1].

Proportional Relationship to CPU Allocation

The memory configurations are directly proportional to the CPU resources allotted to a Lambda function [1]. This relationship emphasizes the importance of choosing the correct memory configuration for the required processing power and effectiveness.

Cost Implications of Memory Allocation

The chosen memory configuration directly affects both the execution costs and the performance of Lambda functions. Engineers must balance optimizing performance and ensuring cost-effectiveness since memory allocation is connected to the total cost in the AWS Lambda pricing scheme [2]. By utilizing the AWS price calculator, engineers can acquire more insights into the financial impacts of different memory settings. Engineers can use this tool to make informed decisions about memory allocation, considering both performance requirements and financial constraints.

Determining Optimal Memory Allocation for AWS Lambda Functions

Introduction to Example Lambda Function

An overview of a Python Lambda function (in Figure 1) that implements the Chudnovsky algorithm is given in this section. The example emphasizes the effect of memory allocation on the

function's performance and cost, providing a practical basis for clarifying the ideas discussed in the paper [3, 4].

```python
import json
from decimal import Decimal, getcontext

def lambda_handler(event, context):
    num_iterations = event['num_iterations']
    pi_estimate = chudnovsky_algorithm(num_iterations)
    return {
        "statusCode": 200,
        "body": json.dumps({
            "pi_estimate": str(pi_estimate),
        })
    }


def factorial(x):
    result = 1
    for i in range(1, x + 1):
        result *= i
    return result


def chudnovsky_algorithm(n):
    getcontext().prec = n + 2  # Set precision

    C = 426880 * Decimal(10005).sqrt()
    result = Decimal(0)

    for k in range(n):
        numerator = factorial(6 * k) * (3 * k + 1) * (3 * k + 2) * (3 * k +
3)      denominator = factorial(3 * k) * factorial(k)**3 * 2**(3 * k)
        result += Decimal(numerator) / Decimal(denominator)

    result = C / result
    return result
```

Figure 1: Lambda Function Implementing Chudnovsky Algorithm

The Chudnovsky algorithm is a fast method for calculating the digits of π, based on Ramanujan's π formulae. The Chudnovsky brothers published it in 1988 [4].

Manual Determination of Memory Settings

Lambda simplifies configuration, allowing for a focus on developing innovative solutions. The memory setting stands out as the primary configuration option influencing performance. Function configuration allows memory adjustments from 128 to 10240 MB in 1 MB increments, with increased memory directly correlating with added CPU power. Refer to Figure 1 to see a Python Lambda function illustrating the impact of memory on performance. Start with a memory setting of 128 MB and set the timeout to 15 minutes.

Execute the function in Fig 1 and take note of the execution duration. Then, increase the memory to 512 MB and rerun the function. What is the new execution time? Further, double the memory to 1024 MB and observe the resulting execution duration. Continue incrementing the memory until reaching a point where additional memory no longer yields performance gains. Identify this saturation point. Applying this strategy to the Lambda Function in Figure 1 and executing with memory settings of 128 MB, 512 MB, 1 GB, 2 GB, 3 GB, and 4 GB, we see the trend shown in Figure 2.

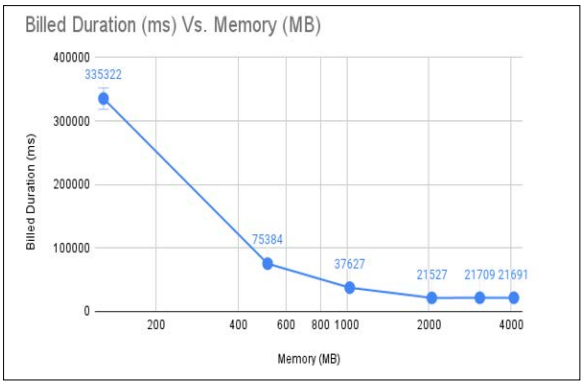

Figure 2: Memory (MB) Vs. Billed Duration (ms) for the Lambda Function Shown in Figure 1

Observation

With each doubling of memory, the execution duration decreases by approximately 50%, illustrating a direct correlation between memory and CPU power. In the case of this application, the breakeven point is around 2GB of memory. Allocating more than 2GB of memory does not enhance performance and incurs costs without corresponding benefits. Fine-tuning the function in Figure 1 may be necessary for further performance optimization.

Power Tuning with AWS Lambda

The process above outlines a manual approach wherein a Lambda function is systematically run with varying memory settings to discern the correlation between memory allocation and performance. Although this approach offers insightful information about the best memory setup for specific functions, managing several Lambda functions in a team setting might be difficult. As organizations maintain numerous Lambda functions to support diverse applications, the manual iteration across different memory settings becomes cumbersome and time-consuming. In such scenarios, the need for a more scalable and automated approach arises, allowing teams to efficiently manage and optimize the performance of many Lambda functions simultaneously. This shortcoming introduces the necessity for tools and frameworks that streamline the process of determining and fine-tuning memory settings across various Lambda functions, ensuring efficiency and ease of management within a team environment.

AWS provides an open-source solution known as Lambda Power Tuning, offering engineers a comprehensive tool to assess the performance of their Lambda functions across different memory settings [5]. This tool empowers engineers to make well-informed decisions regarding the optimal memory configuration, balancing performance enhancements with cost considerations. The functionality of Lambda Power Tuning is facilitated through the deployment of a Step function workflow and a set of Lambda functions specifically designed for testing and cleanup purposes [6]. This robust tool streamlines the evaluation process, enabling engineers to fine-tune their Lambda functions for optimal performance while managing associated costs effectively.

Figure 3 and Figure 4 below show the AWS Lambda Power Tuning Step function workflow deployed in AWS and executing the function for ten iterations at various memory settings.

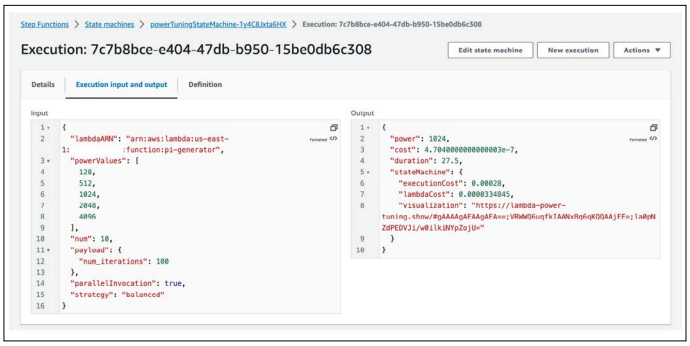

Figure 3: AWS Lambda Power Tuning Execution Result

The results produced by AWS Lambda Power Tuning align with the manual discovery, indicating that the optimal memory setting for achieving the best execution time is 2 GB. This correlation underscores the tool's effectiveness in automating the evaluation process and substantiates the reliability of its output in identifying the most performance-efficient memory configuration.

AWS Lambda Power Tuning offers a variety of configurations, providing flexibility to achieve specific and targeted settings for optimal performance and cost-effectiveness [5].

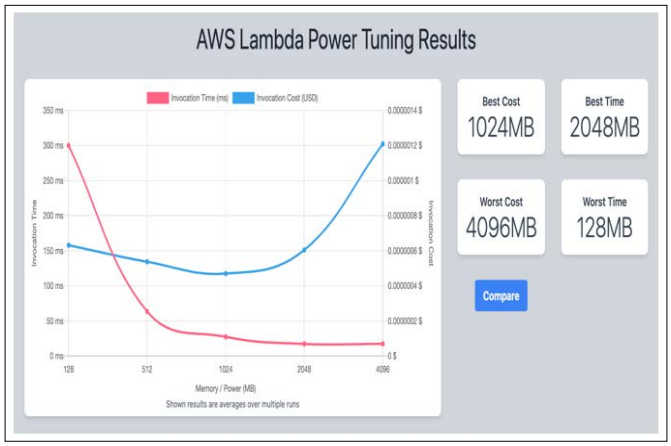

Figure 4: AWS Lambda Power Tuning Results

Conclusion

The critical topics of memory allocation in AWS Lambda functions have been covered in this technical paper, focusing on how memory allocation directly affects costs and performance. We have demonstrated the complex link between memory, CPU power, and function execution time by analyzing the manual determination of memory settings and applying the Chudnovsky algorithm to an illustrative Lambda function example. The investigation illustrated the difficulties of managing several Lambda functions within a team by showcasing the manual method of repeatedly modifying memory and tracking performance improvements.

We introduced and illustrated the effectiveness of the AWS Lambda Power Tuning framework in automating the evaluation process. The best memory configuration for the example program is confirmed to be 2 GB by the tool, which agrees with manual findings. Additionally, Lambda Power Tuning offers a variety of options, giving developers a scalable and effective way to achieve desired settings that strike a balance between cost and performance.

Developers and teams looking to optimize their AWS Lambda functions can use the insights in this paper as a guide. In the world of serverless computing, resource optimization is critical. The increasing need for scalable, economical, and efficient solutions makes using tools such as Lambda Power Tuning essential. By enabling developers to make well-informed decisions, this research advances our understanding of memory use in serverless architectures and eventually improves the efficiency and affordability of serverless applications.

References

1. Memory and computing power. AWS Lambda Operator Guide https://docs.aws.amazon.com/lambda/latest/operatorguide/computing-power.html.
2. Cost optimization. AWS Lambda Operator Guide https://docs.aws.amazon.com/lambda/latest/operatorguide/cost-optimize.html.
3. Chudnovsky David, Chudnovsky Gregory (1988) Approximation and complex multiplication according to Ramanujan. Pi: A Source Book 596-622.
4. Chudnovsky algorithm. Wikipedia https://en.wikipedia.org/wiki/Chudnovsky_algorithm.
5. AWS Lambda Power Tuning. AWS Application https://serverlessrepo.aws.amazon.com/applications/arn:aws:serverlessrepo:us-east-1:451282441545:applications~aws-lambda-power-tuning.
6. Alexcasalboni / aws-lambda-power-tuning. GitHub https://github.com/alexcasalboni/aws-lambda-power-tuning#what-does-the-state-machine-look-like.