Image Processing Pipeline

Java vs. Python Implementation

Aaron Burnham†
Computer Science
University of Washington
Tacoma, Washington USA
atburn@uw.edu

Jose Rodriguez
Computer Science
University of Washington
Tacoma, Washington USA
joser27@uw.edu

Jay Phommakhot
Computer Science
University of Washington
Tacoma, Washington USA
jphomm@uw.edu

An Ha
Computer Science
University of Washington
Tacoma, Washington USA
aha25@uw.edu

ABSTRACT

The Image Processing Pipeline is a comparative study and implementation framework that we designed to evaluate the performance of image processing tasks between programming languages, Java and Python. The pipeline includes six core functions: *image details, rotation, resizing, grayscale, adjusting brightness, and type conversion.* These operations are deployed on AWS Lambda, leveraging its serverless architecture for scalability and efficiency. By using a single AWS account to host all Lambda functions, this study ensures a controlled and fair testing environment.

The conducted study on key performance metrics, including *total cost, function runtime, processing throughput, memory usage, network latency, and cold/hot startup*. A standardized testing methodology is employed, where each language processes identical datasets of images in ten iterations to ensure uniformity in benchmarking. A JavaScript automation script manages the deployment of Maven projects, uploading this snapshot file of functions to AWS Lambda, and then having another JavaScript file to run all the operations, both individually and in a batch where all functions for both languages are run and the aggregated metrics are calculated.

Metrics are collected and analyzed at both granular (function-specific) and aggregate (batch runs) levels. Results from this analysis highlight performance trade-offs between Java and Python, offering insights into their sustainability for various image processing tasks. The study also examines the impact of AWS Lambda configurations, such as memory allocation and runtime environment, on the overall performance and cost efficiency.

This pipeline provides a comprehensive framework for assessing serverless image processing workflows, offering valuable guidance for developers and researchers in selecting the optimal language and deployment strategy for their applications. By emphasizing fairness and reproducibility, this study establishes a benchmark for future explorations in serverless computing and cross-language performance comparisons.

CCS CONCEPTS

Computing methodologies → Parallel computing methodologies → Parallel programming language [*High Relevance]*

Networks → Network services → Cloud computing **[***High Relevance]*

Computing methodologies → Artificial intelligence → Knowledge representation and reasoning → Logic programming and answer set programming **[***Medium Relevance]*

KEYWORDS

Serverless Computing, Image Processing, AWS Lambda, Java, Python.

1 Introduction

This paper presents a comparative study of image processing tasks implemented in Java and Python on AWS Lambda. Our goal is to evaluate the performance and cost-effectiveness of these two popular languages in a serverless environment. To achieve this, we have developed an Image Processing Pipeline that encompasses six core image-processing operations:

1. **Image Details:** Extract metadata such as image dimensions, transparency data, and color depth.
2. **Rotation:** Rotate an image by a specified angle.
3. **Resizing:** Scale an image to a specified size.
4. **Grayscaling:** Convert an image to grayscale.
5. **Brightness Adjustment:** Modify the overall brightness of an image.
6. **Type Conversion:** Convert an image between different formats (e.g., JPEG/JPG, PNG).

Each of these operations is implemented as a separate AWS Lambda function, allowing for fine-grained control and efficient scaling. By deploying these functions to AWS Lambda, we can leverage its serverless architecture to automatically scale the application based on demand, ensuring optimal performance and cost-efficiency.

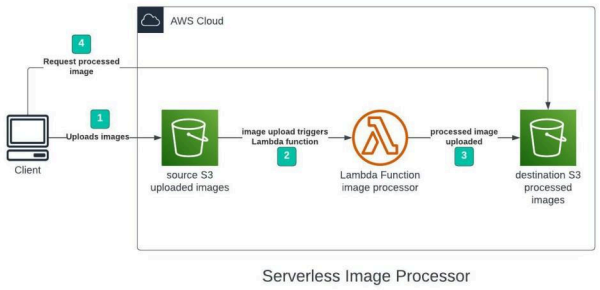

Figure 1: **Serverless Image Processing Flow**

1.1 Research Question

To address the design trade-offs in our image processing pipeline, we will investigate the following research questions:

RQ1: How do performance metrics (runtime, memory usage, throughput) of image processing functions on AWS Lambda vary between Java and Python implementations?

Java and Python have different performance characteristics, such as processing speed, memory management, and resource utilization. By comparing these factors, we can determine which language is more efficient for image processing tasks in a serverless environment.

RQ2: How does memory reservation size on AWS Lambda affect the runtime, throughput, and hosting costs of image processing tasks implemented in Java and Python?

Memory reservation size in AWS Lambda directly influences both the performance and cost of serverless functions. Understanding the impact of memory settings on different languages allows for better cost optimization and resource allocation.

2 Case Study

In this case study, we evaluate the performance of an image processing pipeline implemented in Java and Python. The primary objective is to compare the efficiency and cost-effectiveness of these two languages when deployed on AWS Lambda. The pipeline consists of six core functions, which include extracting image details, performing rotation, resizing images, applying grayscaling, adjusting brightness, and converting image types. These tasks serve as fundamental operations in various image-processing applications, providing a comprehensive basis for performance comparison.

The study was conducted using AWS Lambda with a memory allocation of 1,024 MB. For the Python implementation, the PIL library was utilized, while Java employed the Java ImageIO library. The dataset used for testing comprised three images of varying sizes categorized as small (~50 KB), medium (~500 KB), and large (~5000 KB). Each image processing task was executed ten times for both the Java and Python implementations to ensure reliability and consistency in the collected data. A JavaScript automation script facilitated the deployment and execution of the Lambda functions, enabling uniform testing conditions for both implementations. During each iteration, key performance metrics, including total cost, function runtime, processing throughput, memory usage, and network latency, were systematically recorded.

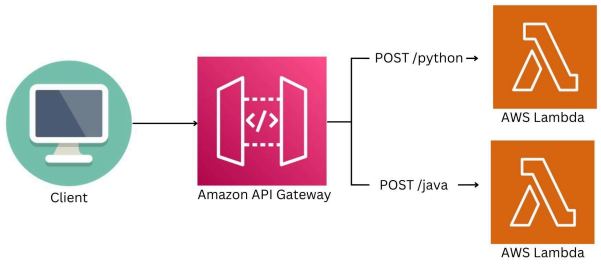

Figure 2: **System architecture for executing the image processing.**

2.1 Design Tradeoffs

Java is a strong choice for compute-intensive tasks that need consistent and predictable performance. It handles large-scale operations well, but its memory usage is higher, and it often takes longer to start, especially in serverless settings like AWS Lambda. This can make Java less efficient for tasks requiring quick response times or minimal resource consumption.

Python, by contrast, is simpler to work with and better suited for tasks that need quick development and execution. Its integration with AWS services is seamless, and its lightweight nature makes it a good fit for dynamic environments. However, Python may incur slightly higher costs for prolonged or intensive workloads. Choosing between the two depends on the project's specific requirements and constraints. The choice between the two depends on the specific needs of the project.

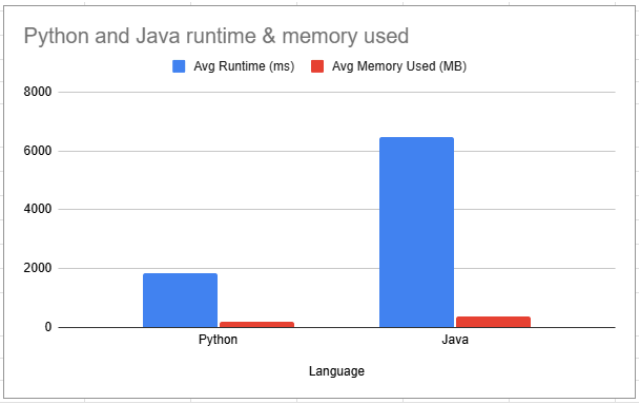

Figure 3: **Average runtime (ms) of both Java and Python**

2.2 Serverless Application Implementation

AWS Lambda and S3 Buckets. This study has leveraged the core characteristics of serverless architecture, particularly through AWS Lambda. Each of the core image functions (Key details, rotation, resize, grayscaling, brightness, and type conversion) are deployed as Lambda Functions. The functions were triggered by a call to the REST API and would access an S3 General Purpose Bucket that holds the testing and output images.

The point of a serverless application is to focus on logic without the necessity of managing servers or infrastructure. AWS Lambda has abstracted this by providing a FaaS infrastructure.

The logic of being able to acquire information about costs, being a *Pay-as-You-Go* service, was calculated during Lambda Function calls. The study bases its location in the *US East (N. Virginia) "us-east-1"* region as it was the location with the lowest cost of *$0.00001667/GB/sec*.

We have been able to integrate other cloud-native services such as API Gateway to allow the AWS Lambda functions to be exposed as a REST API. The study has also allowed the team to utilize AWS CloudWatch in order to monitor metrics such as *runtime, memory usage, and cold start latency*.

This integration of a serverless application has allowed for modularity by independently having each image function as a deployable Lambda function. The functions can be updated or replaced without affecting other components.

We have been able to handle both cold starts (initial loading of the function) and hot starts (subsequent invocations) efficiently. By comparing metrics between Java and Python, we can accurately estimate the trade-offs of whether to use a Java implementation or a Python implementation.

2.3 Experimental Approach

Our first step is to implement the core functions in both Java and Python. Both implementations were designed to handle identical input formats and produce comparable results.

We had a controlled environment by having a single AWS account that deployed all Lambda functions, ensuring consistency in resource allocation and configuration. Lambda functions for both languages were assigned identical memory allocations, timeouts, and environment variables to eliminate variability in runtime conditions. Images of three distinct sizes were prepared to simulate real-world environments. Each dataset contained the same set of images to ensure that both languages processed identical input data.

We used a JavaScript script to automate the deployments of Lambda functions, execution of tests, and the collection of metrics. The scripts handled building and packaging both Java and Python implementations, then uploading them to AWS Lambda which could then be triggered for individual or batch-processing tasks.

Comprehensive performance data was collected, including:

- Runtime (ms): Time from function invocation to completion.
- Memory Usage (MB): Total memory consumption during execution.
- Throughput: Number of images processed per second.
- Cost (USD): Calculated using AWS lambda pricing for runtime and memory allocation.
- Cold/Hot Start Performance: Measured latency differences between the first cold invocation and the subsequent invocations (hot start).

To evaluate the scalability of the serverless application, the study ran batch tests where both Java and Python implementations processed an image from one of three sizes (small, medium, and large) using multiple transformations in a single invocation. Metrics from batch runs were analyzed to assess aggregate performance and average costs.

This studies' testing framework was designed to minimize variability, with all functions triggered under the same conditions and datasets. Logs and metrics were recorded in a structured format for transparency and replicability.

This experimental approach ensures a fair and unbiased comparison between Java and Python implementations, providing actionable insights into their performance and cost-effectiveness for serverless image processing workflows.

3 Experimental Results

Image Access. Using official AWS libraries, our results show that the Python implementation was significantly faster than the Java implementation when downloading images. For images of varying sizes (50 KB, 500 KB, ~5,000 KB), the Python implementation was able to access the image in ~122 milliseconds with low fluctuations between file sizes. However, the Java implementation took a significantly longer time that increased with the file size, averaging ~4,921 milliseconds. The Python implementation here shows a 97.5% better performance than Java.

Function Runtime. We measured function runtime as the total time for the function to complete from invocation. As expected, smaller images had a lower function runtime than larger images for both implementations; however, the Python implementation was consistently performing faster than the Java implementation. Across all functions and image sizes, the Python functions averaged ~1,847 milliseconds, and the Java functions averaged ~6,452 milliseconds.

However, the performance of the image type conversion function was similar for both implementations. This was the only function where the Java implementation would sometimes outperform the Python implementation; all other functions performed significantly faster.

Cost. The cost of each function was estimated using the function runtime above. Using pricing values from AWS, Python functions averaged ~$0.00016 per invocation, while Java functions averaged ~$0.00055 per invocation.

Memory Usage. For memory usage, our findings followed a similar pattern to the other metrics. The Python implementation used an average of ~189 MB for all three file sizes with low fluctuation, while the Java implementation averaged ~378 MB that increased alongside the image size.

4 Conclusions

This study reveals Python as the superior choice when it comes to serverless image processing on AWS Lambda. Python has demonstrated a significantly shorter runtime across small, medium, and large image sizes compared to Java. Small image processing showed a significant difference in runtimes, where Python's average runtime being ~308 milliseconds and Java's average runtime is ~707 milliseconds showing Python outperforming Java by 56%. When processing the largest image, there was an almost four times difference between Python's ~4,562 milliseconds and Java's ~16,715 milliseconds, again showing Python outperforming Java by 72%. This indicates that there could be some potential inefficiencies when using the AWS libraries in Java image processing.

Python also showed a lower memory usage compared to Java, regardless of image size. This could be due to a less optimized way of memory handling. Python also achieved a higher throughput (images processed per second) for small, medium, and large image sizes, which could indicate better efficiency for tasks requiring quick processing. Java's throughput being drastically lower, particularly for larger images could limit its suitability for high-volume, real-time processing.

When taking into account memory reservations on AWS Lambda, Java, while being functional, suffers from higher runtime, increased memory usage, and lower throughput, especially for larger images. These inefficiencies lead to higher costs and limit Java's viability in a serverless environment.

This study establishes a foundation for selecting programming languages in serverless image processing pipelines, emphasizing runtime efficiency and cost optimization. While Python outperformed Java, future research could focus on optimizing Java's image processing libraries and exploring alternatives for both languages to improve performance in serverless architectures.

REFERENCES

[1] Chapin, John, and Mike Roberts. *Programming AWS Lambda: Build and Deploy Serverless Applications with Java*. O'Reilly Media, 2020.

[2] Sharma, Sagar. "Java Vs Python Programming Language - Which is Better?" *Analytics India Magazine*, 1 August 2024, https://analyticsindiamag.com/topics/java-vs-python/. Accessed 11 December 2024.

[3] SnapLogic. (2023, December 14). *Python vs. Java: Head-to-Head Performance Comparison*. https://www.snaplogic.com/glossary/python-vs-java-performance

[4] Lelek, Tomasz, and Jon Skeet. *Software Mistakes and Tradeoffs: How to Make Good Programming Decisions*. Manning, 2022. Accessed 11 December 2024.

[5] Sarkar, Aurobindo, and Shah, Amit. Learning AWS: Design, Build, and Deploy Responsive Applications Using AWS Cloud Components, 2nd Edition. Germany, Packt Publishing, 2018.

[6] GeeksforGeeks. (2023, October 19). *Serverless Image Processing with AWS Lambda andS3.* https://www.geeksforgeeks.org/serverless-image-processing-with-aws-lambda-and-s3/. Accessed 11 December 2024