GW SCHOOL OF ENGINEERING & APPLIED SCIENCE

Assessing Serverless Machine Learning: Predicting Resource Execution Times as a Function of Resource Input

2020 Research under Professor Timothy Wood

openNetVM, George Washington University School of Engineering and Applied Science

Assessing Serverless Machine Learning: Predicting Resource Execution Times as a Function of Resource Input.

openNetVM, George Washington University School of Engineering and Applied Science

Abstract

Serverless computing has become a prominent host of computing resources for proprietary companies due to its flexibility, scaling, and cheap startup costs within Function as a Service (FaaS) tools. Serverless functions such as Amazon Lambda utilize event-driven functions to handle many loads of server and application requests without manual allocation to on-site servers, and can easily scale costs based on required functions. However, we observed that the aforementioned serverless functions, through FaaS, do not differentiate resource execution times based upon the type of function loads or requests. We collected previously tested data on 10 image-based serverless functions, and extracted them from a MariaDB database. In observing these functions as graphs after distinguishing warm and cold container starts, we found that they had high linear correlation, and therefore utilized linear regression models alongside validation testing harnesses. We found that our linear hypothesis largely proved true in our predictions, through the relatively accurate levels of error, but found several varying distinctions of MSE (error) levels between each image function. Therefore, we recommend to the openNetVM integration team of this machine learning research to instead pursue a more practical classification system of predicting blocks of execution times that serverless platforms employ, in order to optimize the university serverless platform.

Introduction

The day and age of advanced computing systems and applications begs a system for a plethora of server hosts and computing power. As many proprietary companies of these aforementioned systems seek to minimize manual operations, server costs, and manual scaling labor -- the advent of cloud computing has become the most popular host of these resources. Serverless computing is a novel platform that provides computer services on a pay-as-you-go basis, so companies only pay for the resources (e.g. memory, CPU, and graphics usage) that they use[1]. This architecture is very different than the traditional system of renting servers to run code because a company needs to predict their resource use beforehand and pre purchase the servers that meet their expected needs, and predictions cost time and money and may end up being inaccurate, causing either service delays due to a lack of resources, or a waste of money due to an overprovisioning of resources. With serverless computing, on the other hand, servers do not have to be reserved, as the service is autoscaling. Servers can automatically be fired upon request, such as in times of high demand, and when this demand goes away, these servers can be shut down and used elsewhere, saving money for companies[1].

On a holistic basis, cloud architectures manifest within several different forms - by the purpose they provide. The first being Software as a Service (SaaS), where a provider-managed client allows the customer or client to retrieve access to the provider's portfolio of applications, and oftentimes, are able to configure a few mechanisms themselves[2]. The second, Infrastructure as a Service (IaaS), allows the end user a plethora of opportunities through gaining access to all features of the provider's platform and applications. Similar to Amazon EC2, the functions supported by the platform -- be it deploying operating systems, applications, or parts of network control -- the user is able to operate[3]. The third, and the focus of this paper's assertions, is Functions as a Service (FaaS). FaaS services are a subset of serverless computing itself, enabling users to execute code and functions without manual and common infrastructure related to microservices. Through FaaS, all related physical hardwaver, virtual machine operating systems, and web server/software management are handled via the cloud provider[4]. FaaS employs a robust cloud infrastructure, allowing a pay by use auto scaling model for proprietary companies hosting through FaaS related services.

While numerous cloud providers provide ample serverless computing platforms, for the sake of this paper, we will predicate our computing analysis and background upon Amazon Web Service's Lambda function, an event-driven serverless platform - one of the most adopted and popular functions of both cloud platforms and AWS itself. Lambda is a service that allows specific functions to run automatically in response to an event without provisioning or managing servers (i.e. "event-driven")[5]. These events may range from adding or modifying an object in an S3 bucket, updating rows in a DynamoDB table, or receiving an HTTP request. Lambda functions automatically trigger to do whatever action is asked or specified, akin to how software is automatic in updating their platforms. Be it automating resources with fault-toleration, scalability in requests, consistency with execution, or its relatively cheap cost, Lambda provides a competitive niche of benefits[5].

While functions such as Lambda handle and scale requests with ease, and scales based upon its need, it does not discriminate this load or scaling request per *type* of function and what the function it achieves on a microservice level. Rather, it classifies based upon collective load. For this reason, we propose a primitive deep learning model that predicts resource execution based upon both categorical classification of the function type, and accounting the resources involved in the function. This research is important because having an accurate estimate of the execution time of a function invocation will lower the number of resources (CPU time or memory) that is assigned to the function. A more accurate execution prediction will also prevent a situation where there's a lack of resources available to functions, avoiding situations where a function's container ends up taking orders of magnitude longer to execute. However, conducting experimentation on these functions can become difficult because Machine Learning requires relevant and numerous features to provide an accurate prediction. If the serverless functions that are on the platform do not use many features in their code, the ML model has less information to use, and it becomes harder to train the model. While we cannot optimize proprietary models such as AWS Lambda as closed-source, we instead aim to use open source serverless platforms such as OpenWhisk to test and run these functions locally.

Methods

Previous researchers invoked 10 image-based serverless functions on OpenWhisk.

1. .Sharp_blur (blur picture):
    a. Sigma blurring coefficient
2. Sharp_sepia (matrix recombination of image)
3. Sharp_resize
    a. Target width
4. Sharp_convert
    a. Format
5. Wand_blur
    a. Sigma blurring coefficient
6. Wand_denoise
    a. Noise threshold

    b. Denoising Softness\

7. Wand_edge (detect edges)
8. Wand_resize
    a. Target width
9. Wand_rotate
    a. Rotation angle (90,180,270)
10. Wand_sepia

The data collected was stored in a MariaDB database, and includes statistics on the run ID, the function, the failed status, start, end, initial startup (init_ms), and wait time for startup (wait_ms), the memory usage for the function, the parameters used for the function, the inputted image and its size (width of the image, height of the image, and file size) for the function, and the output of the function (such as the outputted image size).

Visualizations

Through Jupyter Notebooks, we extracted data from the MariaDB database, and utilized python for data visualization, narrowing features, and regression modelling. This was done through the *Sklearn* library, which included trained models for the data that split 10 files per function. Execution time was taken as the mere difference of start_ms and end_ms, and after transferring SQL data into a DataFrame data structure, there were no observed missing values, as represented in Figures 1-3.

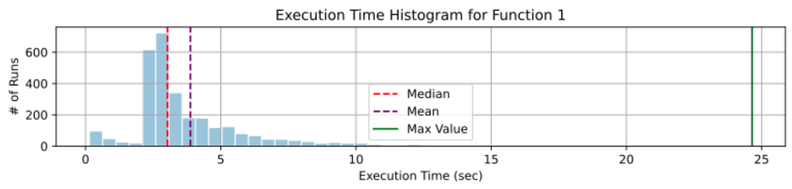

Figure 1. Histogram of Execution Times for Function 1

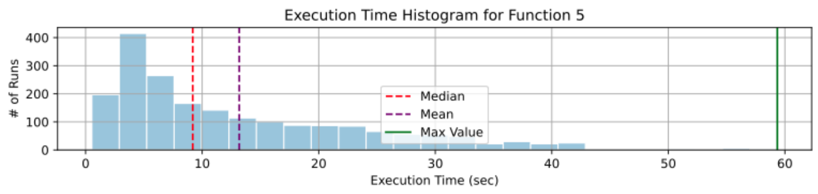

Figure 2. Histogram of Execution Times for Function 5

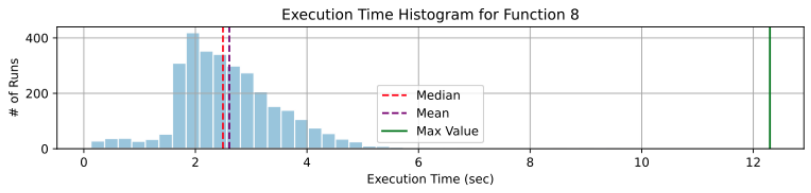

Figure 3. Histogram of Execution Times for Function 8

We noticed was that all the function runs had a right skew and had max execution times that were considered outliers (1.5*IQR) In order to make sure that future modeling would not be skewed by outliers, we decided to remove the outlier runs from the function dataframes, after which we then plotted scatterplots for all the functions to see if there was a relationship between the file size of the inputted file and the execution time. Our initial hypothesis was that the higher the size of the image file, the longer it would take to run the function on the image.

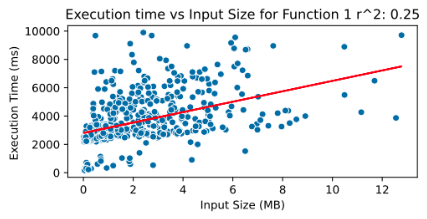

Figure 4. Execution Time as a Function of Input Size for Function 1

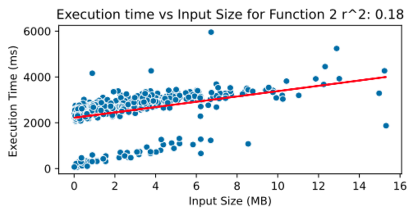

Figure 5. Execution Time as a Function of Input Size for Function 2

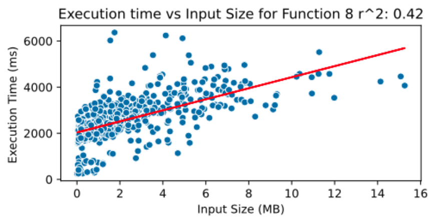

Figure 6. Execution Time as a Function of Input Size for Function 8

In all the scatterplots, we observed a Linear Regression Model was likely most helpful in predicting a function's execution time. Furthermore, we also noticed that for Function 2 (in Figure 5) and Function 3, there was a bimodal split in the execution times, and specifically for Function 2, the split was at approximately 1000 ms. Pursuant to further discussion with our colleagues and Professor Wood, it seemed this occurred due to warm containers (ready to be used) and the others being run on cold containers (that have to be started from origin). Because of the bimodal split, the r^2 performance metric Function 2 correlation was only 0.18, the second-lowest. From this, we included container startup as a selective display.

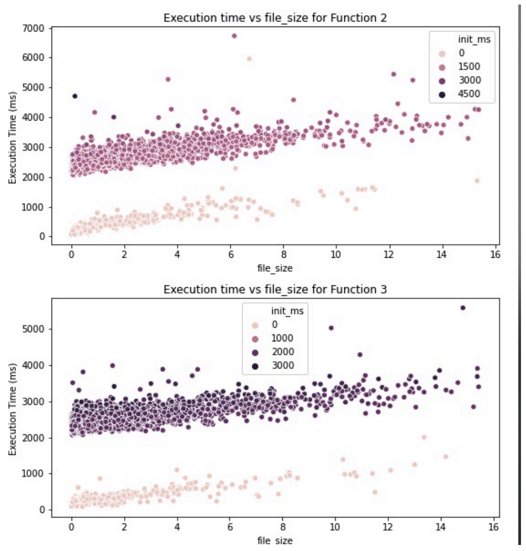

Figure 7. Execution vs file_size for Functions 2 and 3

We graphed each feature of the run against execution time on a scatterplot to see if there was any correlation.

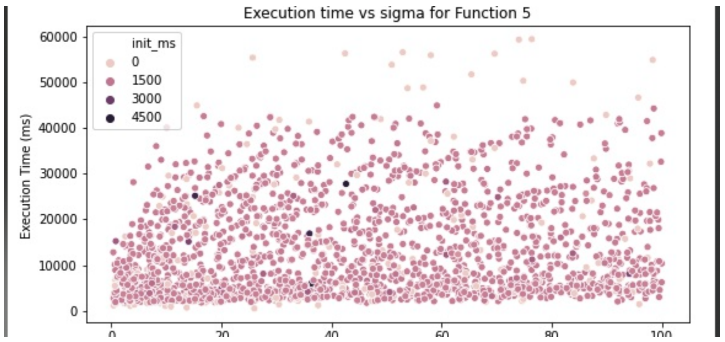

Figure 8. Execution time vs sigma for Function 5

For Function 5 (In Figure 8) and Function 8, there wasn't any observable relationship between the feature and its execution time. For Function 5, execution time vs sigma produced a blob of a scatter plot, and cold vs warm starts weren't separated distinctly like for Function 3. This suggested to us that it was necessary to include all the parameters of a function in any training model we choose and that one feature would not be enough to accurately make a prediction.

Now, we were ready to train our models with the data. In accordance with the aforementioned data analysis, we decided to make a new feature isWarm for each function run, which related to the aforementioned discussion on containers.

Performance was evaluated by comparing the Mean Squared Error (MSE) of the model, where a lower MSE indicates predictions closer to the true value. Therefore, minimizing the MSE would be our goal. However, in *Sklearn*, the MSE is reported in negation, meaning that scores can be reported as ascending: where the largest score is the best. We first confirmed the ability of isWarm to lower the error of the model by comparing the MSE of using and not using the isWarm feature on a Linear Regression model.

For all the functions, there was a decrease in MSE. There was a higher drop in MSE for the functions that had a clear split between warm and cold runs. However, for Function 5, there was only a small decrease in MSE without run separation, meaning we needed to test other models.

We tested LinearRegression, Lasso, Ridge, ElasticNet, KNeighborsRegressor, DecisionTreeRegressor, RandomForestRegressor, SVR regression models for all the functions below.

Results

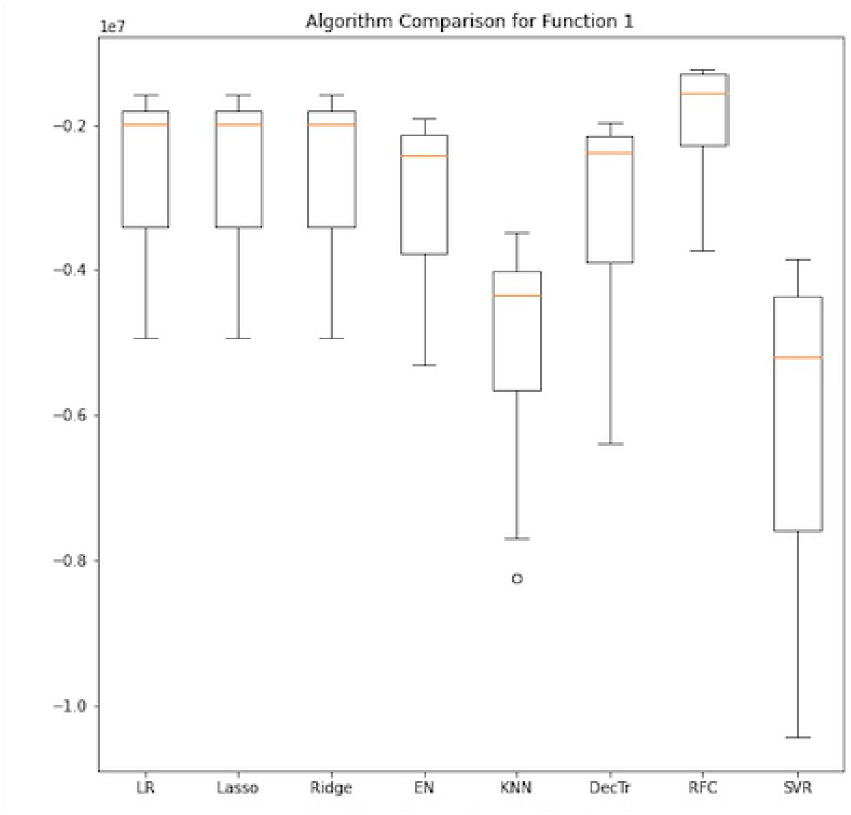

Figure 10. Algorithm Comparison for Function 1

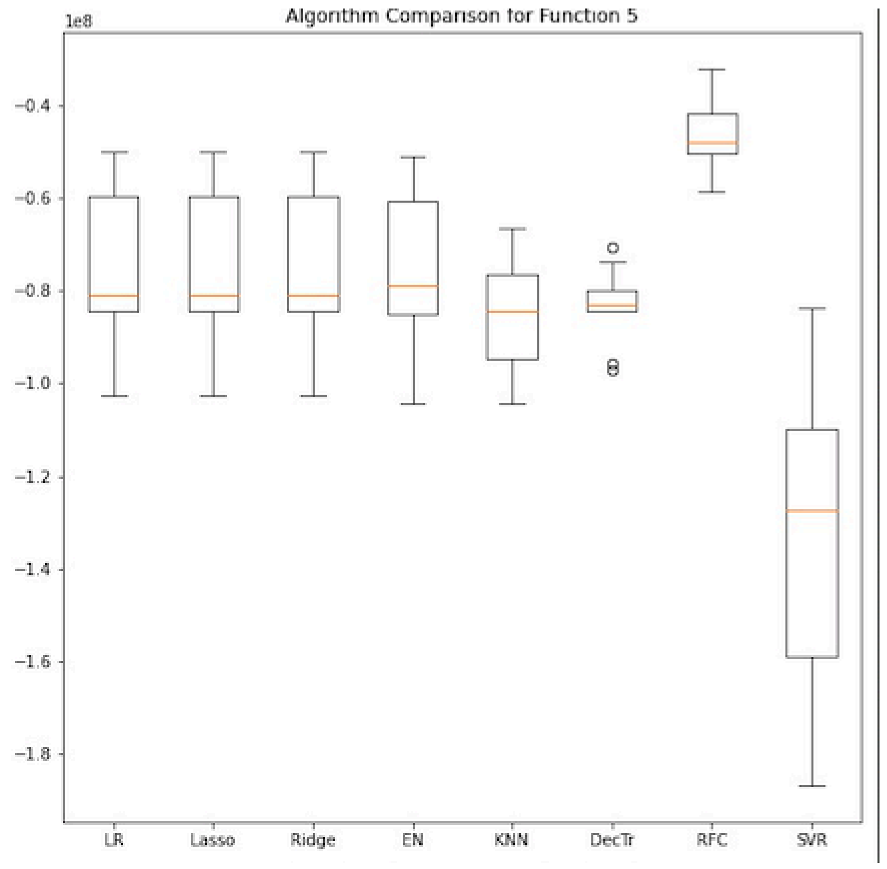

Figure 11. Algorithm Comparison for Function 5

Consistently for all the Functions, we noticed that Linear Regression, Lasso, Ridge, and Elastic Net Regressions had very similar errors because they are all deviations of linear models - meaning that they do not differ by much.. While Functions 1 and 5 were more optimized, we noticed Function 5 had MSE was in the range of tens of millions. Although this was lowest among the models and MSE was reported as a negative value (meaning the largest value indicated the lowest error), it was still a very high number. Furthermore, we knew that image features and execution times were not highly correlated.

Discussion

In retrospect, linear models were genetically more reliable in simple execution time prediction, given most of the other features didn't have much correlation anyway. Container splits in Functions 2-3 were optimized by Random Forest Classifications, which are fast in training and are relatively easy to make predictions from - which is why most serverless functions use these models to execute in mere milliseconds. The Mean Squared Error differed widely in terms of orders of magnitude. For function 2, the MSE was hundreds of thousand before Function 5's. The linear model was not constant, because predicting execution time as a *singular* time through linear regression is not favorable - likely because the model finds difficulty in generating a classification of one specific time. In the future we would instead wish to experiment predicting a "block" of execution time instead of a singular time. For production serverless platforms, the usual scale is 100 to 500 ms, meaning users are charged for use every 100 to 500 ms, which makes for a *much* simpler modeling approach because there are fewer things to predict. This facet also changes the problem into a categorical prediction instead of a regression prediction, opening up more models that could be tested aside from the constraints we utilized.

References

1. Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2019. The rise of serverless computing. Commun. ACM 62, 12 (December 2019), 44-54. DOI: https://doi.org/10.1145/3368454
2. “What Is Saas? Software as a Service: Microsoft Azure.” *Software as a Service | Microsoft Azure*, azure.microsoft.com/en-us/overview/what-is-saas/.
3. Sitaram, Dinkar and Geetha Manjunath. “Infrastructure as a Service.” CloudCom 2012 (2012).
4. By: IBM Cloud Education. “Faas.” *IBM*, www.ibm.com/cloud/learn/faas.
5. Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. Peeking behind the curtains of serverless platforms. In 2018 USENIX Annual Technical Conference (USENIX ATC 18), pages 133–146, Boston, MA, July 2018. USENIX Association.

*Referenced as superscript in text citations.