
# Distributed Deep Learning Model on Serverless Architecture

**Ayush Sharma (ams2581)**
**Soamya Agrawal (sa3881)**

**2021**

# Table of Contents

# Abstract

Due to rapid growth of data both in volume and variety, there has been need to scale up machine learning which in turn has sparked broad interests to develop distributed machine learning systems, typically based on parameter servers. Most of the current Deep Learning training jobs use data parallelism where large number of training samples are processed by different workers in parallel. Combined with cloud storage, data parallelism aligns with the serverless architecture naturally, where user can launch a number of stateless functions, each accessing a different batch of data. In serverless architecture, there is no need to manage any dedicated servers or instances for their applications. Hence, with the serverless architecture, resources can be easily scaled up or down by controlling the number of functions launched at any point. In serverless computation, a user only needs to pay for the execution time his/her functions is active. Serverless compute is the cheapest form of compute offered by any cloud providers. Due to its lightweight nature, ease of management, and ability to rapidly scale up, distributed serverless computation has become the trend of building next-generation web services and applications.

**Keywords:** Serverless, AWS Lambda, Distributed, Deep Learning

# Chapter 1

# Introduction

We implemented Distributed Deep Learning model on Serverless Architecture and compared it with Traditional Deep Learning Architecture on time, accuracy, and cost metrics. To train distributed Deep Learning architecture, we have used data parallelism and serverless functions (AWS Lambda). To compare it with traditional Deep Learning architectures, same model was trained on the entire dataset on GPU as well as CPU. For all these approaches, we compared time, train/validation accuracy and cost.

This helped in studying the benefits of distributed serverless architecture. We learned that serverless architecture is easy to scale up/down based on the requirement. There is no overhead to maintain any server. It is cost effective as user pays only for the execution time of serverless functions, and that is cheap. Since data parallelism is used, for huge datasets, training time reduces by a significant amount.

## 1.1 Background Work

Following the footsteps of traditional virtual machines and Docker containers, serverless (or Function-as-a-Service) architectures, represented by AWS Lambda and Google Cloud Functions, have emerged as a burgeoning computation model to further reduce costs and improve manageability in cloud computing. AWS Lambda was used to implement serverless function. Google Cloud Platform(GCP) was used to implement traditional Deep Learning architecture on GPU as well

as CPU. Tutorials on how to invoke AWS Lambda using Boto3 and how to invoke AWS Lambda functions in parallel from Parameter Server code written in Python were helpful to implement distributed serverless functionality. To implement deep learning model on AWS Lambda, we referred to a couple of tutorials to help us in importing TensorFlow and Keras as AWS Lambda layer.

## 1.2 Technical Challenges

There was not any reference code for distributed serverless architecture implementation. Hence, we implemented the entire distributed Deep Learning serverless architecture from scratch.

We started implementing distributed architecture using JavaScript as it is inherently asynchronous, but it did not have support for matrix calculation needed for weight aggregation. Hence, we had to switch to Python. Python has support for matrix calculations but is not inherently asynchronous. Hence, it was a challenge to invoke AWS Lambda functions in asynchronous manner to achieve distributed functionality.

There was a learning curve to learn AWS Lambda and also to import libraries like Tensorflow and Keras as layers in AWS Lambda.

The total unzipped size of the AWS Lambda function and all layers can't exceed 250MB. Due to this constraint, we were not able to import tensorflow_datasets library along with Tensorflow, Keras. Due to the above memory constraint, every AWS Lambda function had to load the entire dataset, and then partition it according to the offsets provided for training.

# Chapter 2

# Approach

## 2.1 Overview

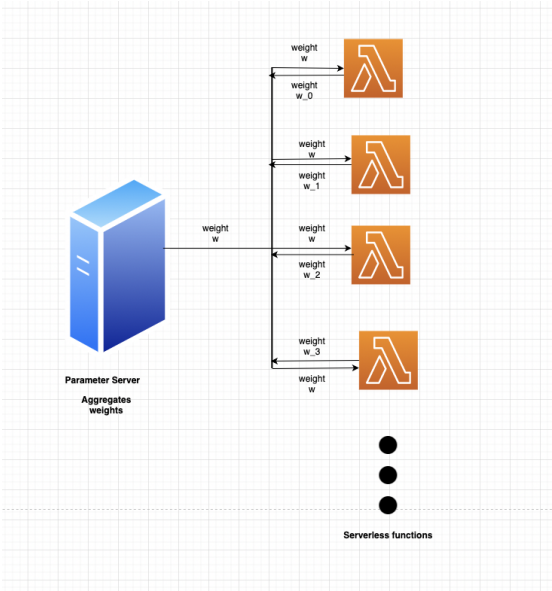
We implemented a deep learning model to train CIFAR-10 dataset. The dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class, 50000 training images and 10000 test images. To find out the train/validation accuracy, cost and execution time for traditional Deep Learning architecture, the above dataset was trained on GPU (offered by Google Cloud Platform) and CPU (offered by Google Cloud Platform).

To achieve Distributed Deep Learning Architecture on serverless architecture, we used AWS Lambda as serverless functions, which were invoked in parallel using Python code. To implement data parallelism, CIFAR-10 dataset was divided equally into batches of training instances based on the total number of AWS Lambda functions used. Corresponding AWS Lambda functions were invoked. After every epoch, parameter server aggregates all the weights and sends the updated weights to be used for the next epoch. The above approach was varied across the different number of AWS Lambdas. We then noted down the training time, train/validation accuracy, cost for the deep learning model ran on GPU, CPU and different number of AWS Lambdas.

Since AWS Lambda has memory limit, we had to create a model with less number of trainable parameters, and hence we could not use models like ResNet-50. After getting all the above information, we plotted corresponding graphs to drive easy comparison and reach a conclusion.

## 2.2 Solution Architecture

Since the model replicas are trained using different input data, model parameters will typically diverge. To reconcile these parameters and ensure that all model replicas eventually converge, each replica periodically pushes its set of parameter values to a centralized server, called the parameter server. The parameter server aggregates all the received updates for each parameter and then sends back to all replicas the newly computed set of values, which is used at the beginning of the next iteration. The workers pull the model from the parameter servers, compute on the Deep Neural Networks(DNN), and then send the computed gradients to the parameter servers.




## 2.3 Implementation Details

Dataset: CIFAR-10 dataset

Serverless function: AWS Lambda.

Number of AWS Lambdas: 1, 5, 10.

Number of epochs for which training was done for all the scenarios: 100

Training data was divided equally in batches depending on the total number of AWS Lambda functions.

GPU instance details: 4 vCPUs, 15 GB RAM NVIDIA Tesla P100 x 1

CPU details: 4 vCPUs, 15 GB RAM

Deep Learning model:

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 32, 32, 32) | 896 |
| conv2d_1 (Conv2D) | (None, 32, 32, 32) | 9248 |
| max_pooling2d (MaxPooling2D) | (None, 16, 16, 32) | 0 |
| conv2d_2 (Conv2D) | (None, 16, 16, 64) | 18496 |
| conv2d_3 (Conv2D) | (None, 16, 16, 64) | 36928 |
| max_pooling2d_1 (MaxPooling2 | (None, 8, 8, 64) | 0 |
| flatten (Flatten) | (None, 4096) | 0 |
| dense (Dense) | (None, 32) | 131104 |
| dense_1 (Dense) | (None, 10) | 330 |

Total params: 197,002
Trainable params: 197,002
Non-trainable params: 0

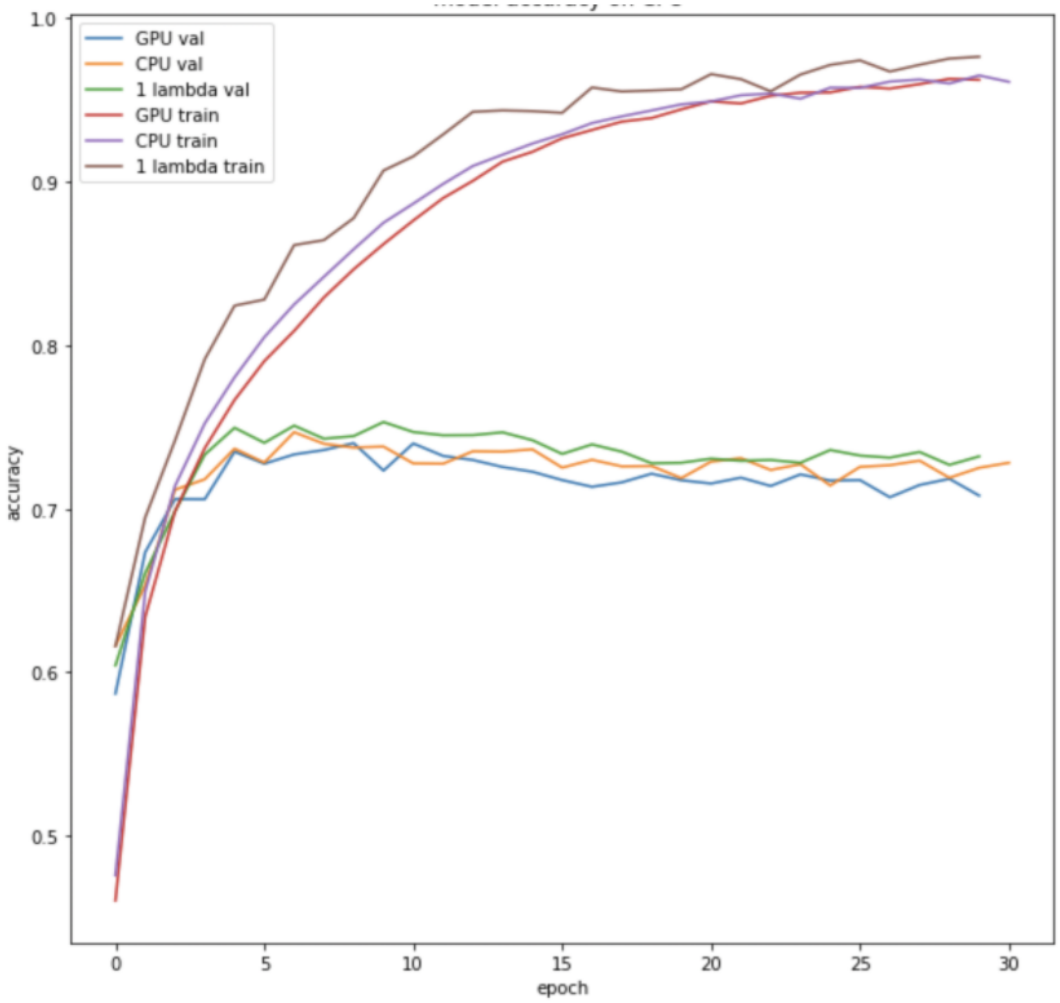
# Chapter 3

# Experimental Evaluation

1. AWS Lambda caches dataset explaining the reduction in execution time for subsequent calls to AWS Lambda functions.

```
      loss  accuracy  sparse_categorical_accuracy  epoch  billed_time  execution_time
0  2.110566   0.20854                      0.20854    1.0   456.646155       54.740601
time for  epoch 1  is :-  54.74060106277466  seconds
b.py:70: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this,
' when creating the ndarray
  weights=np.true_divide(weights,lambdas)
      loss  accuracy  sparse_categorical_accuracy  epoch  billed_time  execution_time
0  2.100427   0.21238                      0.21238    2.0   212.348818       27.263649
time for  epoch 2  is :-  27.263648986816406  seconds
```

2. Comparing train/validation accuracy




Average epoch time(in sec):

GPU: 6

CPU: 150

AWS Lambda: 200

3. Cost per epoch(in $):

GPU: 0.00193

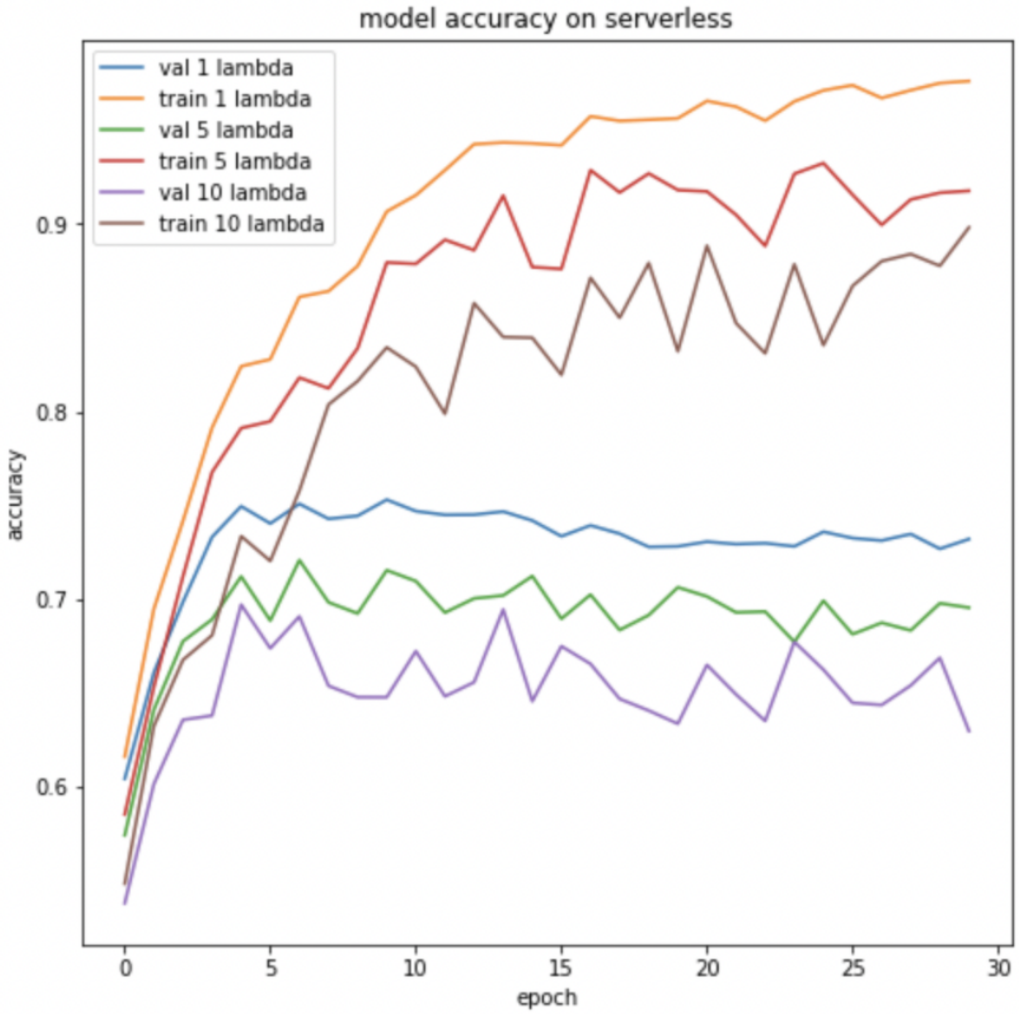
CPU: 0.00587

AWS Lambda: 0.00334

* Lambda beats CPU in cost by being 40% cheaper and a optimal architecture as discussed later can beat the CPU in time as well.

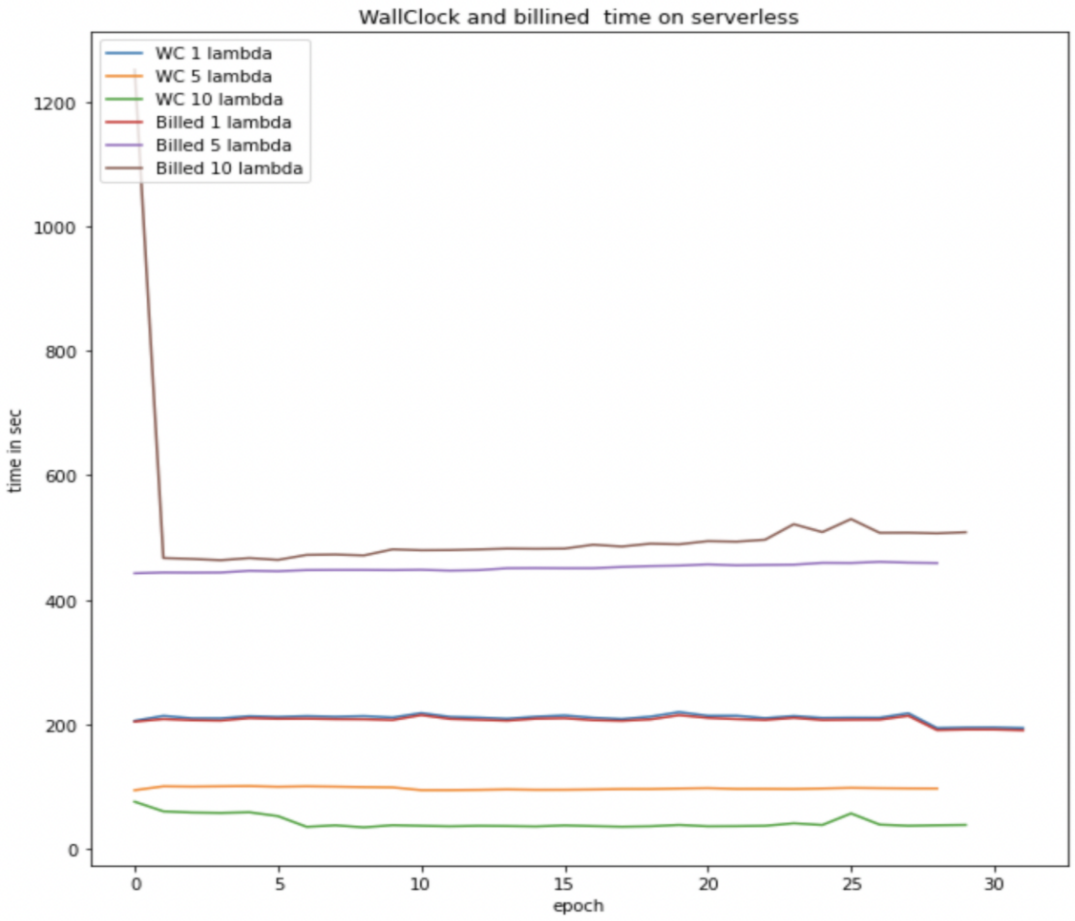
**In GPU and CPU, user pays for the duration of the VM, AWS Lambda pays for exact compute time only.

4. Comparing train/validation accuracy by varying total number of AWS Lambda functions




* While Increasing the number of AWS Lambda functions, we get a comparable train and validation accuracy. (It is less that than running on 1 AWS Lambda since it is batch training.)

5. Comparing wall clock time and billed time by varying total number of AWS Lambda functions




*Doubling the number of AWS Lambda functions gives a great reduction in the wall clock time, hence showing that lambdas are inherently scalable.

# Chapter 4

# Improvements

While implementing distributed Deep Learning model on serverless architecture, we realized that improvements could be made as part of this.

1. Since we are using data parallelism, in our implementation, each AWS Lambda function loads the entire dataset, and then train the model on subset of the data based on the offset specified by the parameter server. This increases the time taken by the AWS Lambda function. tensorflow_datasets library could be used to load the data that is required.
2. Rather than implementing a relatively simpler model to train dataset due to memory constraint of AWS Lambda, state-of-the-art models like ResNet-50 could be used by implementing a common database that could be accessed by all the AWS Lambda functions.
3. In our implementation, the number of AWS Lambda functions invoked is same for every epoch. Deep Reinforcement Learning could be used to dynamically scale the number of AWS Lambda functions required to optimize cost/accuracy.
4. To achieve good accuracy as part of the distributed serverless architecture, it is important to shuffle the data, so that AWS Lambda function doesn't get trained on subset of target classes.
5. Asynchronous and scalable nature of serverless architecture can be easily exploited for asynchronous distributed training.

# Chapter 5

# Conclusion

1. Serverless architecture is easy to scale up and down. There is no additional time required to set up new servers, as we ran the experiment for various total number of lambdas by just changing the parameter value.
2. Time required to train on GPU is lesser than time required to train on CPU or AWS lambda but it is not scalable.
3. The Elasticity of AWS Lambda gives great power to dynamically scale, an efficient architecture suited for the serverless could give better results in terms of time and cost both compared to traditional CPU.
4. Accuracy achieved by all the above cases is more or less similar.
5. Serverless Distributed also opens up avenue for Cost optimal ML algorithms.