

# A Framework to define optimized Shuffling process to reduce data transfer latency: MapReduce in Serverless Infrastructure

MSc Research Project  
MSc in Cloud Computing

Achyut Anantakumar Vadavadagi  
Student ID: x18131557

School of Computing  
National College of Ireland

Supervisor: Manuel Tova-Izquiere

National College of Ireland  
Project Submission Sheet  
School of Computing



<b>Student Name:</b>	Achyut Anantakumar Vadavadagi
<b>Student ID:</b>	x18131557
<b>Programme:</b>	MSc in Cloud Computing
<b>Year:</b>	2019
<b>Module:</b>	MSc Research Project
<b>Supervisor:</b>	Manuel Tova-Izquiredo
<b>Submission Due Date:</b>	12/12/2019
<b>Project Title:</b>	A Framework to define optimized Shuffling process to reduce data transfer latency: MapReduce in Serverless Infrastructure
<b>Word Count:</b>	6151
<b>Page Count:</b>	21

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

**ALL** internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

I agree to an electronic copy of my thesis being made publicly available on TRAP the National College of Ireland's Institutional Repository for consultation.

<b>Signature:</b>	
<b>Date:</b>	10th December 2019

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:**

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
<b>Attach a Moodle submission receipt of the online project submission</b> , to each project (including multiple copies).	<input type="checkbox"/>
<b>You must ensure that you retain a HARD COPY of the project</b> , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

<b>Office Use Only</b>	
Signature:	
Date:	
Penalty Applied (if applicable):	

# A Framework to define optimized Shuffling process to reduce data transfer latency: MapReduce in Serverless Infrastructure

Achyut Anantakumar Vadavadagi

x18131557

MSc Research Project in Cloud Computing

## Abstract

Today in cloud computing, serverless platform is appeared as a modern technology with the combination of Function as a Service (FaaS) and Backend as a Service (BaaS) with significant computational power. The FaaS is the class of serverless computing service in which user has a privilege to deploy their functions with no management of hardware's by the user. It enable the users to run multiple tasks concurrently with higher elasticity and scalability by introducing fine-grained billing. Cloud providers adds resources arbitrary to deploy and run functions by millisecond level billing. MapReduce is framework or model for decentralized processing of data, where it is broadly operates for processing the large data-sets. This paper considers MapReduce in serverless platform where shuffling creates the latency issues because the object doesn't scale due to Input Output per second limitations during processing of data-sets. This paper provides the Composite Model for processing large data-sets defined in greater performance serverless platform by executing the tasks on Function as a Service (AWS Lambda) and Backend as a service (bringing combination of fast storage and slow storage). An evaluation has been carried out to understand the suitability of MapReduce in serverless platform and experimentation is carried out and compare with other platforms. The results shows that AWS lambda is suitable for processing the large data-sets with less execution time with moderate billing.

# 1 Introduction

## 1.1 Brief History

In 1950s, John McCarthy and others introduced the concept of cloud computing where software computing resources and commodity hardware are used to deploy and deliver the user services through internet (Namboori (2019)). By understanding the needs of the customers, cloud computing offers:

- 24 \* 7 availability
- Easily Manageable
- Flexibility in capacity
- Efficiency and so on

With the evolution of cloud, the serverless computing came into existence which provides greater advantages like monitoring, server instance selection, deployment, security patches, logging and so on. Accordingly this infrastructure helps to ease the authorized user to movement their operational responsibilities to the cloud providers which expands the user revolution and liveliness. Serverless infrastructure is the combination of Backend as a Service (BaaS) and Function as a Service (FaaS) where it offers cloud storage services like object storage, in-memory storage, key value database and so on as a BaaS and platform to write functions (AWS Lambda) as a FaaS. Here users write functions and these functions are triggered automatically.

In the current scenario data comes from different place with various types where it is important to analyse the data for the predictions. To analyse the different kinds of data, different techniques came into existence where MapReduce is most popular framework. MapReduce is the engineering model for distributed and parallel processing for the big data. A MapReduce task is to divide the data set into independent pieces which deals by Map jobs in a absolute parallel format. The technique classifies the output of the map function, the classified output acts as input to reduce functions. Here both input and output are stored in the file system (Namboori (2019), Goddard (2018), SeasiaInfotechBlog (2019)).

## 1.2 Evolution of MapReduce in Serverless Environment

In past decades MapReduce is used to run on Hadoop environment which operates with in-parallel large number of nodes which is dependent on the size of the data in turn which causes Latency issue, caching is limited, slow for the processing of large data set and restricted up to batch processing.

The recent trend is set to MapReduce in serverless environment to overcome the issues shown in figure[1]. However running analytics on serverless platform leads to unproductive allocation of data between the jobs i.e., latency issues in the data shuffling phase. This can be achieved by proper fine grained coordination and providing the adequate storage for the fine grained operations. The selection of storage medium possess direct influence on cost and latency. Examining the Map and Reduce functions with N shuffling, which generates  $N * N$  files on cloud based storage. If Amazon S3 or Dynamo DB as storage which impact the poor performance, where as selecting cache in memory

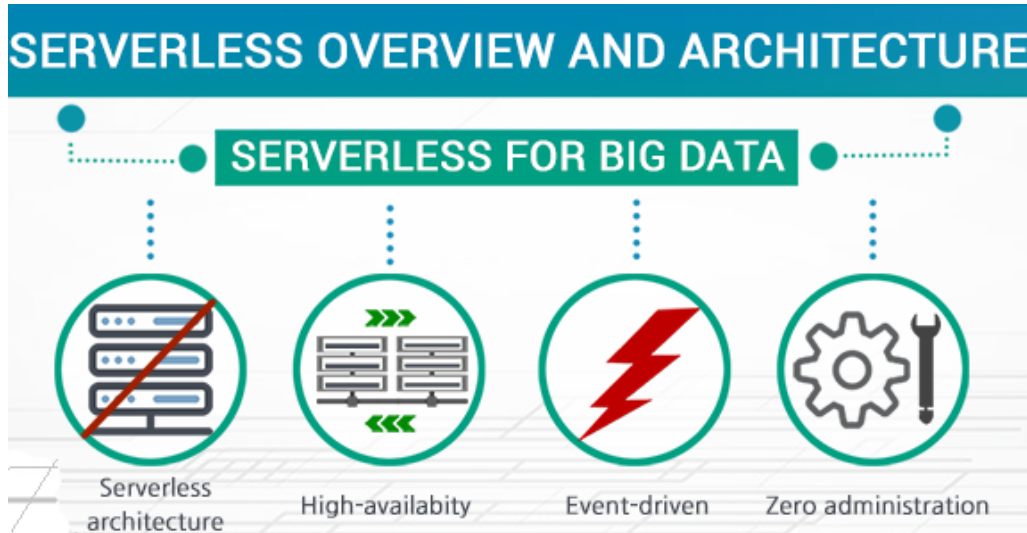


Figure 1: serverless Infrastructure, Source: Xenonstack (2018)

Redis as the storage which provides good performance with the increase in cost (Dashbird (2018), Engdahl (2008)).

In this paper, we define a Composite Model that helps to achieve fine grained elasticity in consideration of cost and performance. We explored the different cloud based storage’s to assist the execution of MapReduce jobs.

### 1.3 Motivation

The implementation of MapReduce in serverless environment by Marla model defines the high latency during shuffling phase. This paper aims at providing the fine grained elasticity to achieve low latency with consideration of the cost.

### 1.4 Research Question

The main objective is to reduce the latency issues during the processing of large data sets in serverless infrastructure. Analysing and finding the limitations of the architecture and employ fine grained elasticity is aimed to achieve in this proposal. Hereby, this research solves the below question:

What are the minimal necessary architecture changes to overcome the difficulties of processing data analytics on serverless infrastructure to improve latency issues by employing the fine-grained elasticity?

### 1.5 Report Structure

The section 2 explains the execution of MapReduce in various environment. It describes the individual platforms with defects, solutions and their outcomes. The section 3 describes the steps, required materials, description about the sample data and measurements of the application. The section 4 describes the Composite model, proposed model, environment setup and code structure. The section 5 and 6 describes the implementation and evaluation of Composite model respectively. The section 7 gives the conclusion and future work of the Composite model.

## 2 Related Work

The section gives the design artefacts, models and test cases of MapReduce in different environments. The table 1 summarize the research areas with pros and cons.

### 2.1 Defects and Provided solutions in MapReduce Hadoop

Sharma and Singh (2018) and Raj and Babu (2015) addresses the problem of data locality of MapReduce which is caused due to network latency. This paper discusses the characteristics of different algorithms apprised of data locality in the process of scheduling. The scheduling algorithms include Delay scheduler, data placement, NKS, Joint scheduler, Job scheduling, and so on. By considering the issue, Cheng et al. (2018) proposed self learning model Resource and deadline aware Hadoop based job scheduler considers the resource availability to reduce the job deadline and estimates the completion time. Dar-tois et al. (2019) also provides solution by utilizing data placing strategy, QoS controller and quantile method. The result shows the green power for the prediction, scalability and accuracy of the job scheduling.

Guo and Agrawal (2018) develops the performance model to notify the minimum requirement from developers to achieve high productivity and surpass Spark noteworthy. It includes three functional operations, map, shuffling and reduce which makes ease to develop middle ware and programming. The experimentation has done by allocating Intel Xeon quad core processor with operational frequency of 2.6GHz including infinite band interconnect of 40GB and 12GB RAM. By experimentation it concludes that 30% of average reduction using MapReduce like API and ten times faster than Spark execution.

Samadi et al. (2018), Cui and Wu (2018) and Kumar et al. (2016) proposes mechanism for addressing the failures of Hadoop MapReduce by improving fault tolerance. For the experimentation, each Job tracker and Task tracker is 3 core with 2 GB RAM Virtual machine. They run Hadoop with version 2.7.4 along the alignment of Centos Linux. One node act as the master node and all other nodes as slave nodes. By experimentation it states that response time of the MapReduce is relying on category of failure and failure rate of job which is directly proportional to response time of the job.

Nabavinejad and Goudarzi (2017) suggests the virtual machine (VM) selection with smart configuration that solve problems of processing big data and provides the greater accuracy, better performance, energy consumption and reduce cost. For investigation, they applied to make span minimization algorithm to explain the effect of this approach. For experimentation they used the MapReduce based PUMA benchmark and describe the adjacency, grep and so on. They conclude that examining more resource and time in profiling phase provides greater accuracy in resource management with high performance.

### 2.2 Investigation on MapReduce Hadoop environment

Kaniwa et al. (2017) Alipour et al. (2016) and Ahmad et al. (2014) calculates the execution time and investigated on performance of defined tasks which is facilitated in Hadoop group of nodes. This paper uses Hadoop platform for running MapReduce jobs using suffix tree data structure. For experimentation, they used Python 3.5.2, Hadoop 2.6.1 and Anaconda 4.1.1. Input and output values are gathered on the distributed file system. The experimentation result states that the MapReduce processing time is excessive in composite distribution systems. By consideration of above issue, Clemente-Castelló et al.

(2018) combines synthetic bench-marking with analytical modelling which helps resolves the overhead occurred by fragile connection at the fine granularity of map and reduce period. Huang et al. (2016) states that the in-memory storage i.e., Alluxio framework empowers the well organization data sharing with moderate outcomes. The fundamental outcomes demonstrate that may accomplished four times speedup than Spark.

Hsaini et al. (2018) defines the architecture which is dependent on Conformance testing to check a Mapreduce implementation under MR-IUT test which is adjust to its particular. This approach uses distributing testing rules which tells how to tackle the problems occurred in MapReduce distributed testing. This helps to improve the security and performance of MapReduce complex framework.

Wei Zhang et al. (2013) made an investigation on Hadoop and MapReduce objectives which refines the capabilities of virus scanning by creating the multi pattern contesting Aho-Corsick methods. MapReduce takes text input format which is re-conceived input type so that any data can be read in binary style. For reducer, to solve same matching query, which is affected by read ahead in the parts, where key and value are made exclusively same. By experimentation, parallel scanning can enhance the capability of virus scanning.

Matthews and Leger (2017) defines the two step anomaly detection technique that uses MapReduce model to process the raw phasor measurement units(PMU) data in Hadoop environment. They implemented the approach on multi-core system to operate the large data-set extracted from PMU carrying approximately 19 million measurement. For experimentation they executed the large data-set on temporal and constraint detection algorithms. The result states, this approach is able to identify the formed anomalies with their behaviours which motivates the MapReduce approaches for SCADA applications.

Wasi-ur Rahman et al. (2013) proposed the blueprint of RDMA (remote direct memory access) based MapReduce Hadoop across the InfiniBand with in-memory integration mechanism for reduce function, data shuffle across InfiniBand and pre-pick-up data for map function. It consist of RDMA listener and receiver, data request queue and RDMA copier. For experimentation, the cluster is composed of Intel Westmere cluster processor with compute nodes operate at 2.7GHz.. The result shows the enhancement in performance which is efficient mechanism for processing the data.

## 2.3 Different proposed shuffling methods

Li et al. (2013) analyse shuffling issues in phase of shuffle job, rebuild shuffle as service, and surpass input output scheduling on map face. The evaluation method is carried with simulation experiment, environment setting and MapReduce job comparisons. By consideration of above issue, Nicolae et al. (2017) finds solution by considering load balancing of data shuffling by utilising the executor level reciprocity, source responsiveness prioritization, and so on. Yu et al. (2015) derived a novel on virtual shuffling technique to qualify well organized data movement and reduce I/O of data shuffling in MapReduce framework. Their experimentation result states that the memory utilization and performance can be improved by accelerating data development in MapReduce.

Daikoku et al. (2018) investigated on decoupled shuffle architecture, coupled shuffle architecture and skew aware meta shuffle with decoupled shuffle architecture ( DSA with SMS) to address the problems occurred by skewed data in MapReduce shuffling phase. Considering the skew tolerance and weak scaling performance the result states that DSA with SMS is the valid solution to tackle the issue.



## 2.4 Investigation on MapReduce Serverless environment

Bardsley et al. (2018a) and Baldini et al. (2017) made a inspect on serverless biological system in a low latency and high accessibility profile setting for analysing the performance of serverless architectures. They bind the examination explicitly to one part of the AWS usage of serverless called as AWS Lambda. Their outcomes appears that there are chances to tune the performance attributes of Lambda based structures and their diagram contemplation, considering potential latency and cold starts qualities made by a integration of elements incorporating events and external frameworks.

Moczurad and Malawski (2018) propose a methodology towards serverless infrastructure that firmly incorporates the Luna programming language and visual-textual with the model of computation. They created intuitive API by incorporating the Luna features and libraries using AWS Lambda to trigger the external functions built in JavaScript.

The advantage of using MapReduce in Serverless environment is cost and performance. Werner et al. (2018), Kim and Lin (2018) and Elgamal (2018) evaluates the scalability, performance and cost effectiveness of serverless computing for big data processing. For experimentation they considered the complex calculations and result shows serverless computing can lower the infrastructure and operational cost with higher performance and scalability.

There are many providers which offers the serverless platform. In order to select the suitable provider, McGrath and Brenner (2017), Bardsley et al. (2018b) and Pelle et al. (2019) evaluated different platforms in high availability with low latency. The evaluation is done by considering the throughput and scaling in their implementations. The experimentation's is done based on the Concurrency test and the Backoff test and result shows AWS Lambda exhibits the highest throughput and better performance which scale linearly at all conditions.

Giménez-Alventosa et al. (2019) designed the platform to execute the MapReduce tasks in serverless environment built on AWS Lambda and AWS S3. They carried out an assessment and result reveals that the non uniform performance behaviour that may threaten for tightly coupled evaluated jobs. Jonas et al. (2019) describes the limitations of MapReduce in serverless environment. This is due to IOPS limitations and data stores latency hence shuffle doesn't scale which results in reduced performance with greater latency. They described that the problem is due to poor standard communication patterns and absence in fine grained elasticity. They have experimented on 100TB of big data for sorting. The result states that sorting for provided data is same as virtual machine instances with rise of cost. By considering the cloud storage issue, Klimovic et al. (2018) investigated on cloud based distributed cache and object storage services to check the suitability as remote storage for serverless environment. Their analysis helps to design the ephemeral cloud storage system which provides the cost efficiency and better performance. The experimentation result states the throughput is major considerable than latency which flash storage gives the stability in cost and performance.

Based on provided solutions, Pu et al. (2019) provided the serverless analytic platform called Locus which guides the selection of storage which varies by performance and cost. They combined the slow and fast storage by analysing the bandwidth and throughput limitations. Their experimentation result shows there is improvement in performance in comparison with the Apache spark environment.

<b>Title of the Papers</b>	<b>Pros</b>	<b>Cons</b>
A framework and a performance assessment for serverless MapReduce on AWS lambda.	Implementation of MapReduce in serverless by AWS Lambda and S3 services.	Solution has latency and cost issues in implementation without considering fine grained solutions.
Cloud programming simplified: A Berkeley view on serverless computing.	Explained about the Serverless computing and limitations, also given solutions to achieve fine grained elasticity.	Provides solutions to latency but the implementation of MapReduce in serverless has more cost.
A big data MapReduce framework for fault diagnosis in cloud-based manufacturing.	Solution that provides decreased cost and improved quality about pattern recognition by MapReduce in Hadoop environment.	This paper doesn't specify the challenges that affects the performance of the classifiers.
A review on data locality in Hadoop MapReduce.	Provided solution for data locality of MapReduce which is caused due to network latency.	Explained about the dependent factors but there are no techniques to tackle them.
Model driven performance simulation of cloud provisioned hadoop mapreduce applications.	It helps modified to simulation prototype that enlarge queuing prototypes to bear nested queuing networks, which is acceptable for performance prototyping of composite distribution systems by MapReduce Hadoop.	The process has approach that as data size increases, execution time also increases. There is no solution for latency.
Parallel algorithm for indexing large dna sequences using mapreduce on hadoop.	Provided solution of mine patterns using MapReduce Hadoop with consideration of performance.	Doesn't specify about the comparison with other platforms like spark which has better performance due to in-memory computation.
Improving the shuffle of Hadoop MapReduce.	Provided shuffling process which is more efficient than traditional MapReduce shuffling.	No explanations about fault recovery and data management which effects the shuffling process.
Leveraging adaptive i/o to optimize collective data shuffling patterns for bigdata analytics.	Provided solution for scalability and performance issues occurred in the shuffle data transfer phase for big data analytics.	Since it is not well organized, which creates problem of conserving energy and power consumption in data movement.

Table 1: Research Areas with pros and cons

## 3 Methodology

### 3.1 Steps

Initially this paper tried to implement the MapReduce in openstack but, the compute node creation took a huge time (large latency) before execution of tasks. By extending the MARLA model, the Amazon Web Services (AWS) platform is used for performing the MapReduce tasks. Although AWS provides the Elastic MapReduce (AWS EMR) service, it doesn't provide the management console and the latency is displaying output is high. The collection of other services like S3 object storage, Redis cache storage and Lambda platform for adding functions provides flexible services. The AWS S3 acts as the trigger event to execute the map and reduce functions. The Cloud Watch service provides the logs which is used to calculate the execution timings, memory allocation, memory consumption and other dependent factors.

The AWS lambda is attached with Redis external library to support fine grained elasticity. The external library is created in EC2 instance and added to S3. The lambda adds external libraries in layers by S3. The layers is functionality of lambda which helps for adding external libraries for the execution of lambda functions.

### 3.2 Materials and equipment's

Using an Amazon Web Service account (AWS), this paper dives into the MapReduce tasks by adding permissions through AWS IAM. The figure 2 explains the different policies need to configure and attach to role before creating lambda function.











Policy name ▾	Policy type ▾	
▶  <a href="#">AmazonElastiCacheFullAccess</a>	AWS managed policy	✘
▶  <a href="#">AmazonS3FullAccess</a>	AWS managed policy	✘
▶  <a href="#">CloudWatchFullAccess</a>	AWS managed policy	✘
▶  <a href="#">AmazonElasticFileSystemReadOnlyAccess</a>	AWS managed policy	✘
▶  <a href="#">AmazonS3ReadOnlyAccess</a>	AWS managed policy	✘
▶  <a href="#">AmazonVPCFullAccess</a>	AWS managed policy	✘
▶  <a href="#">AmazonElasticFileSystemFullAccess</a>	AWS managed policy	✘
▶  <a href="#">AWSLambdaVPCAccessExecutionRole</a>	AWS managed policy	✘
▶  <a href="#">AWSLambdaRole</a>	AWS managed policy	✘
▶  <a href="#">CloudWatchEventsFullAccess</a>	AWS managed policy	✘

Figure 2: Policies used in IAM

The Lambda service is used to execute Map and Reduce functions. The lambda function is created and configured with IAM role, trigger object, network topology, security groups and run-time that needs to be executed. The role (with attached policies shown) has been attached to get all required permissions, the Python with latest version 3.8 has been added as runtime and the network topology has been created by adding following components.

- **Subnets:** The public and private subnets are created by adding the IP V4 address and with port number 20. This helps to delimit the IP address to reside in the VPC.
- **Creating NAT Gateway:** This allows to create the bridge between the public subnet to private one. The NAT gateway is created in the public subnet which helps to access Route Table.
- **Route the subnet in NAT gateway:** This helps to access the internet for our AWS resources. By editing the route table of the subnet which helps to target the new NAT address.
- **Security Groups:** It provides the security at the protocol and port level which filters the incoming and ongoing traffic.
- **Timeout and Memory:** The default timeout is 3 seconds and memory allocated is 128 Mega bytes. This model sets timeout to 9 seconds and 192 memory due to addition of external libraries.

The AWS S3 bucket is created to store input csv file and the reduce function output. The S3 acts as the trigger event to invoke the lambda function. The AWS elastic cache Redis is created to store the intermediate data where data comes from map function and output to reduce function.

### 3.3 Sample data

The data used for MapReduce function is SFI gender based data. Here we perform the operation of number of occurrences of program name present in the comma seperated file(CSV). The map function generates the key value pair by collecting the data and seperates the required and non required data for reduce function. The reduce function collects the key value pair data and outputs the number of occurrences.

### 3.4 Measurements

Due to limitations of resources, the size of data choosen for execution in MapReduce is 6MB. In according to size of data, the lambda functions are created by allocating below memories and timeout.

- **Mappers:** The two mapper functions are created and assigned with 9 seconds timeout and 192 Memory with all network configurations.
- **Coordinator:** A single coordinator function is created and assigned with a minute timeout and 200 Memory with Redis dependencies.
- **Reduce:** A single reducer function is created and assigned with 3 seconds timeout and 128 Memory with Redis dependencies.

## 4 Design Specification

The Marla model (Giménez-Alventosa et al. (2019)) proposed the architecture of MapReduce in serverless platform as shown in figure 3. This model uses the lambda as compute engine and S3 as the storage for input, output and intermediate data. This model wages the latency issue due to absence of fine grained elasticity.

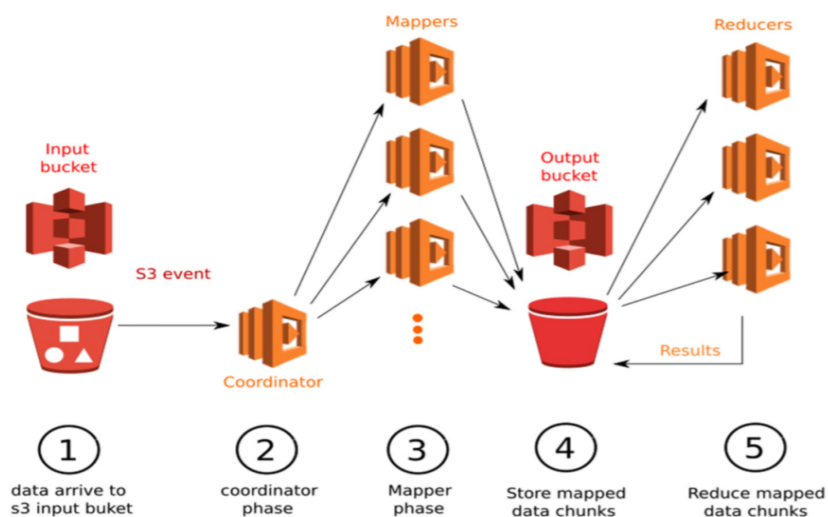


Figure 3: MARLA Architecture

In this section we define the composite architecture to execute the MapReduce tasks using Amazon Web Service resources. The AWS Lambda service works with Amazon VPC (virtual private cloud) through Amazon Direct Connect Link which allows us to perform the MapReduce in the composite architecture as shown in figure 4. By default the AWS is not configured with the VPC, the following components are created.

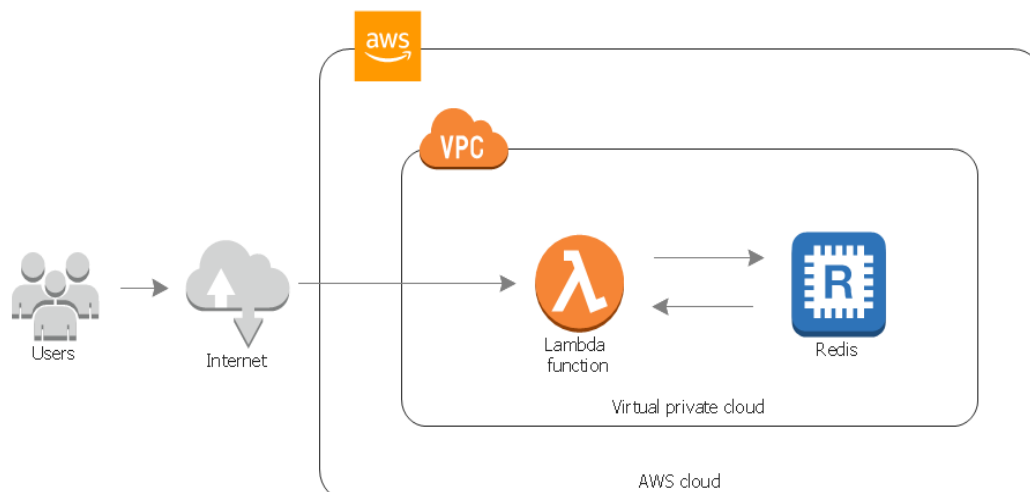


Figure 4: VPC Architecture for Lambda and Redis

The Composite architecture has three phase task execution: read, compute and write. The following section describes the components where the task is executed.

- **Input and Output:** The AWS S3 (Simple Storage Service) is object storage used to store the input data which is need to be manipulated to map function and output data of the reduce function. S3 helps to retrieve and restore the every version of data easily. The redis cache storage is used to store the intermediate data generated from map functions. It collects data from map function and pushes to reduce function.
- **Compute:** The lambda function is used to perform the map and reduce task on the provided data. The lambda function is connected to s3 object storage at input and output phase and redis cache storage for shuffling phase.

The figure 4 represents the architecture which is built in the virtual private cloud (VPC) to provide internet access to lambda and redis services. The VPC is built with private and public subnets, NAT gateway and security group.

The architecture is built by AWS services in absence of external services. We define the architecture to overcome the latency issue in shuffling phase by introducing the fine grained elasticity. The input is txt or csv file stored in s3, intermediate data is stored in AWS Redis and output is stored in s3. It is composed of coordinator, map and reduce lambda functions which defines the basic MapReduce function. The coordinator organize the input data which ensure all the input data is for manipulation and also helps to trigger the map and reduce lambda functions. The map function takes chunks of data which is required for the manipulation. The reduce function sorts all the data and stores the data into S3.

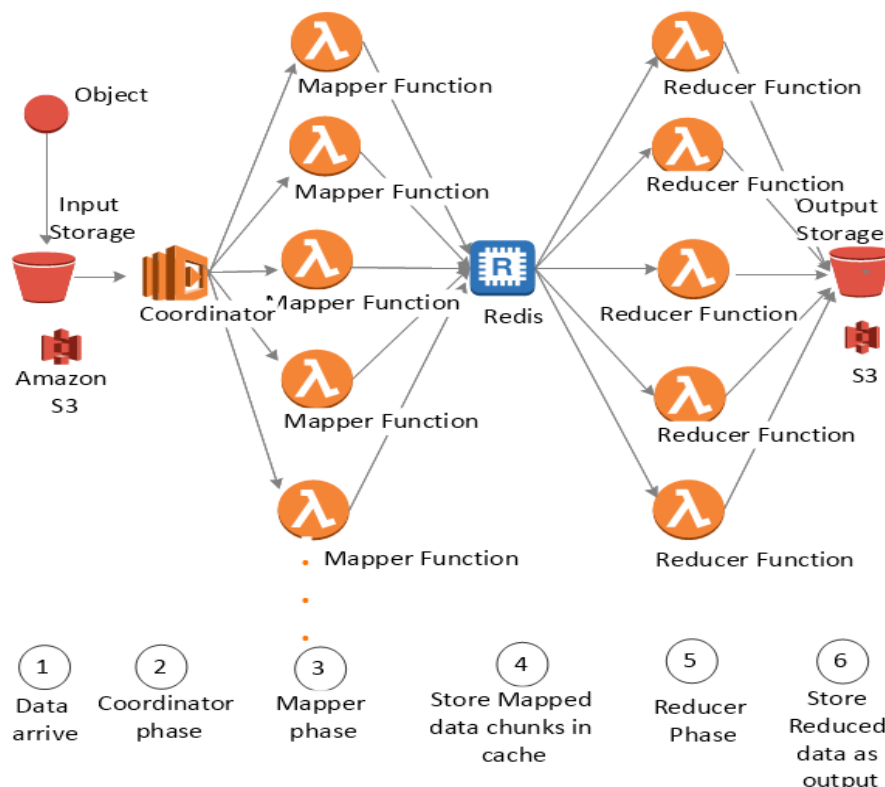


Figure 5: Composite Architecture

Figure 5 depicts the end to end Composite architecture which defines the workflow by using AWS services. The MapReduce in serverless is defined by 6 steps where workflow starts from input phase and ends with output phase.

- **Input phase:** The input data of csv or txt file is added into the s3 which acts as the event to trigger the coordinator lambda function automatically.
- **Coordinator phase:** The given input data is divided into chunks with defined size and ensures all the given data is used in map phase. This stage also helps to trigger the map function where the function is defined.
- **Mapper phase:** The map function split the input data into chunks and further it is used for the processing. This function returns the name-value pairs, a list of 2D tuples in the form of Pair[i][j].
- **Shuffling phase:** In this stage the intermediate data from map function i.e., key value pairs are sorted and stored data is used for the reduce function.
- **Reducer phase:** This function reduces the key value pairs which takes a sorted list as the argument and returns the two dimensional tuple with set of name and value.
- **Output phase:** This phase stores the output of reduce function which is in the form of key and its number of occurrences as the value.

when the object is added to input storage the framework is initialized. This makes s3 event to trigger the coordinator function which calculates the size of data chunks according to availability of memory and RAM assigned to X number of mappers. The coordinator will recompute the size of data piece if map function doesn't have sufficient memory.

In comprehensive with size of data chunk processed by each map function, the coordinator function run the below tasks. First, it calculates the suitable size of each data piece by dividing the total data size available as input to number of map function allocated by the user.

x = number of mappers  
y = total data size (input)

then, dataChunk =  $y / x$

If size of dataChunk is smaller than minimum block size that need to be executed, then the data piece size is counted to size of minimum block size that needs to be executed and number of mappers is calculated as below lb = LeastBlockSize = 1024KB (as example)

$x = \text{int} ( y / \text{lb} ) + 1$

If dataChunk size is greater than the defined fixed memory size (fixedMemorySize), the coordinator function, will figures the fixedMemorySize as percentage of memory allows to mapper lambda function. This helps the data piece to hold by mapper functions. In such scenario the data piece size value is set to fixedMemorySize and x will be recounted.

f = fixedMemorySize

$x = \text{int} ( y / f ) + 1$

If dataChunk size is greater than the defined peak memory size (amount of memory the processor has used), then the value of dataChunk is set to peak memory size and x is recounted.

p = peak memory size

$x = \text{int} ( y / p ) + 1$

## 5 Implementation

We have implemented the Composite Architecture by extending the MARLA algorithm Giménez-Alventosa et al. (2019), a python based MapReduce framework developed in serverless computing. MARLA helps to implement the MapReduce functions that perform data shuffle (intermediate stage) between lambda functions, which lacks the fine grained elasticity. This paper elevate MARLA with assist for shuffle functionality and define performance model that depicts the mechanized shuffle operations. For the composite architecture we utilize the AWS services like AWS S3 as input and output storage, AWS Redis as intermediate data storage and AWS lambda as computation engine.

To execute the MapReduce functions in composite architecture, we define map and reduce function in python3.8 as runtime by importing the library Boto3 for introducing S3 and Redis services. Also, libraries like importing csv for reading csv files and importing Redis for extending the cache memory functionality. Note the fine grained elasticity is introduced by adding functionality of Redis, which is described in section 4 above.

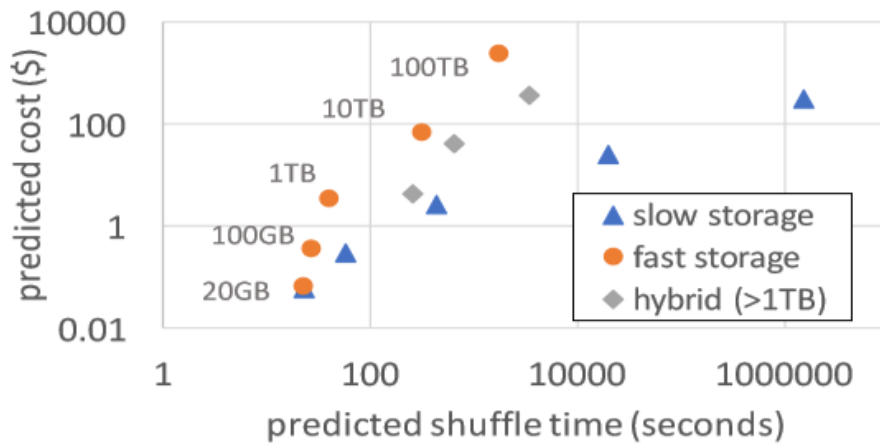


Figure 6: Cost and time comparison for S3 and Redis storages, Source: Pu et al. (2019)

- Model Extension:** The previous shuffle scenario is using S3 object storage for storing the input, intermediate and output files. The S3 for shuffle phase creates the latency issues which is described in figure 7. The cache storage redis considered as fast storage which minimizes the latency between map and reduce lambda functions. Considering the latency issue, the Composite architecture defines the S3 as input and output storage and Redis as intermediate storage defines the efficient performance model.

We have implemented have coordinator, map and reduce function in a multiple lambda functions which is triggered by the S3 input bucket. The role for creating lambda function has full access permission to S3, Elastic cache, VPC, cloud watch and so on as shown in figure [2].

## 6 Evaluation

Composite Serverless model can reduce execution time by up to 30% than Marla model, and the same time being close to AWS EMR completion time up to 2X. Even with



the small amount of fast storage, Composite model helps to reduce the latency in turn improves the performance. The below graph 7 shows the time taken by lambda to execute the different size of data.

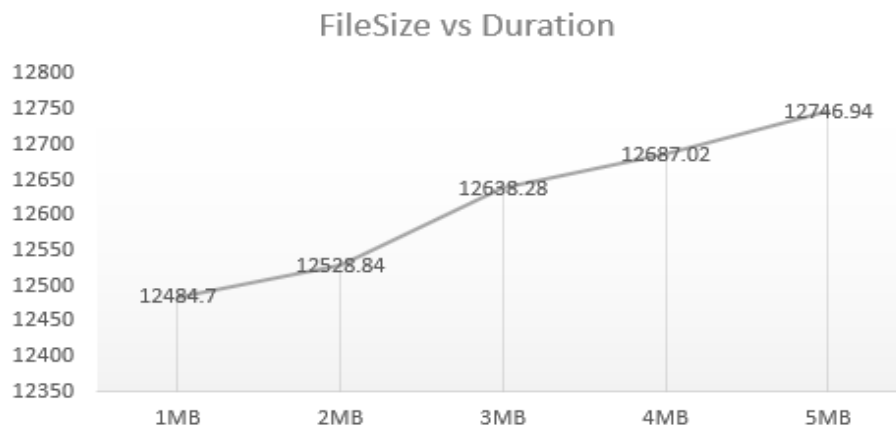


Figure 7: File size v/s time taken for execution

## 6.1 Case Study 1

The file size with 1 MB is chosen to validate the memory usage and cost occurrence. Here this paper creates the test file which is in the form of JSON. After execution, the duration shows the time taken for the execution. Since it is executed successfully, the image 8 shows the output is displayed in green color with the output.

```

Code SHA-256
MIPnn+RJKdUC2xROGkym7Trufo1CDNY4TCd15Ru+ydU=
Request ID
86a45766-cd0a-4dbe-8edb-2c76b32541d0

Duration
12484.70 ms
Billed duration
12500 ms

Resources configured
704 MB
Max memory used
96 MB

Log output
The section below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. Click here to view the CloudWatch log group.

START RequestId: 86a45766-cd0a-4dbe-8edb-2c76b32541d0 Version: $LATEST
FileSize = 0.9673042297363281 MB
chunk size to small (0 bytes), changing to 1024 bytes
numberMappers-- -1023.0326957702637
chunk size to large (1024 bytes), changing to 460 bytes
END RequestId: 86a45766-cd0a-4dbe-8edb-2c76b32541d0
REPORT RequestId: 86a45766-cd0a-4dbe-8edb-2c76b32541d0 Duration: 12484.70 ms Billed Duration: 12500 ms Memory Size: 704 MB Max Memory Used: 96 MB

```

Figure 8: The output for executing 1 MB

## 6.2 Case Study 2

The file size with 11 MB is chosen to validate the memory usage and cost occurrence. Here this paper creates the test file which is in the form of JSON. Since the file chosen is larger than the specified size (6 MB), the image 9 shows the output is displayed in red color with the log.

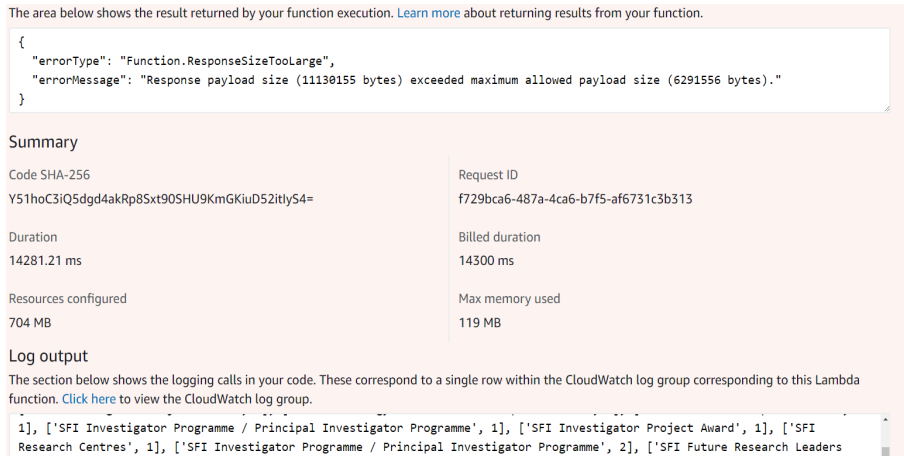


Figure 9: The output for executing more than defined execution file size

Here file size from 1MB to 5MB are executed in Composite model where the completion time is around 12 seconds for all size of data. The latency of 12 seconds is due to cold start, the lambda and redis is built in VPC. The lambda utilize resources from VPC using elastic network interface (ENI), this creates the initial cold start.

The graphs for duration, error count, throttle time and invocation time are provided by monitoring service in AWS Lambda. The duration graph 10 shows the time from when the coordinator function starts executing the given data as a consequence of invocation to when it stops invoking another function. Here the green dot represents minimum duration of the lambda function is executed. The graph 11 shows the error and success rate of execution. The graph has green dot of 100% shows all the data being executed with out any bugs.

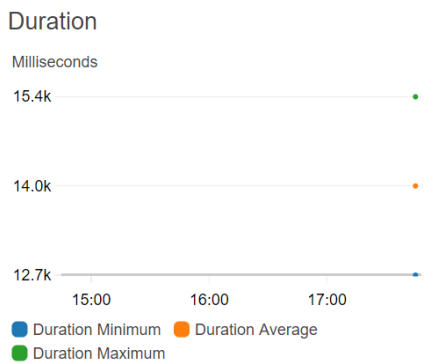


Figure 10: Duration for 5MB

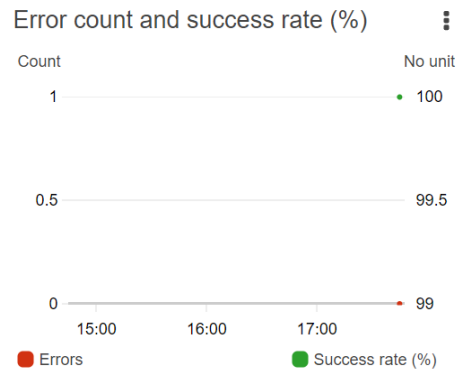


Figure 11: error count for Composite model

The graph 12 shows number of times the function is being called. Here the coordinator lambda is invoked by S3 event and response event from mapper lambda hence it shows 2 times. The graph 13 estimates the coordinator function invocation strive that were throttled anticipated to invocation rates exceptional the given concurrent limitations. The graph shows 0 throttle rate.

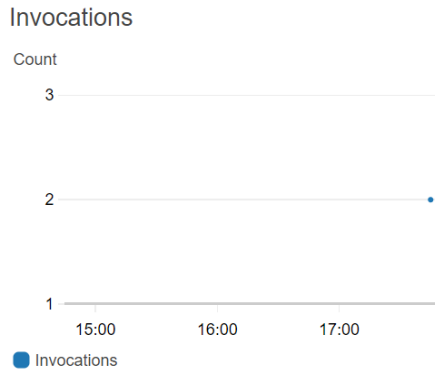


Figure 12: Invocation time for 5MB size

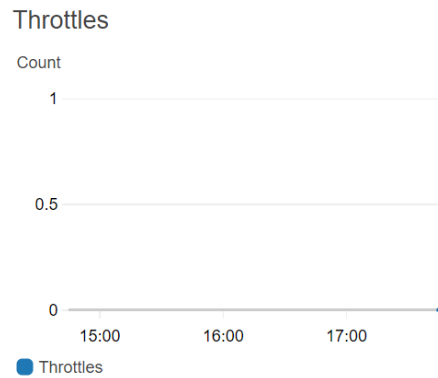


Figure 13: Trottle time

Further extends with the redis, the figure 14 shows the cache memory outline. The below diagram shows the replication factor for the Redis storage. Replication Bytes describes the size of memory that the master is forwarding to all of its replicas. Here the two replicas has been used inside Redis elastic cache.

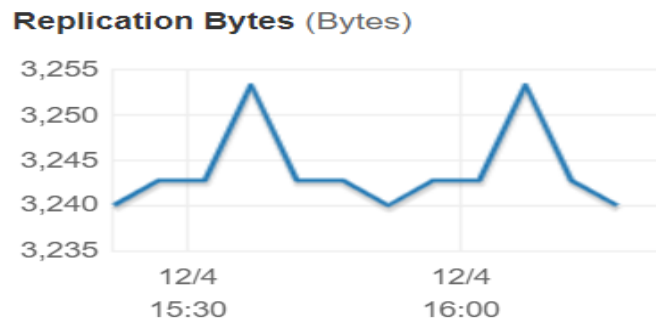


Figure 14: Replication factor for Redis

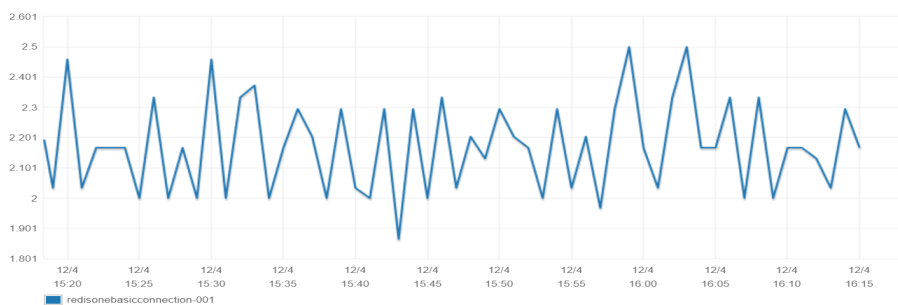


Figure 15: CPU utilization

The graph 15 helps to monitor the workload. The graph shows that the usage is under the threshold value that is under 45%. Here the default threshold value has been set to 90%. The graph 16 shows the packets in and out for the in-memory computing which has related with the network bytes in and out. This represents the data in and out from Redis has equal size of data where 100% data has been utilized.

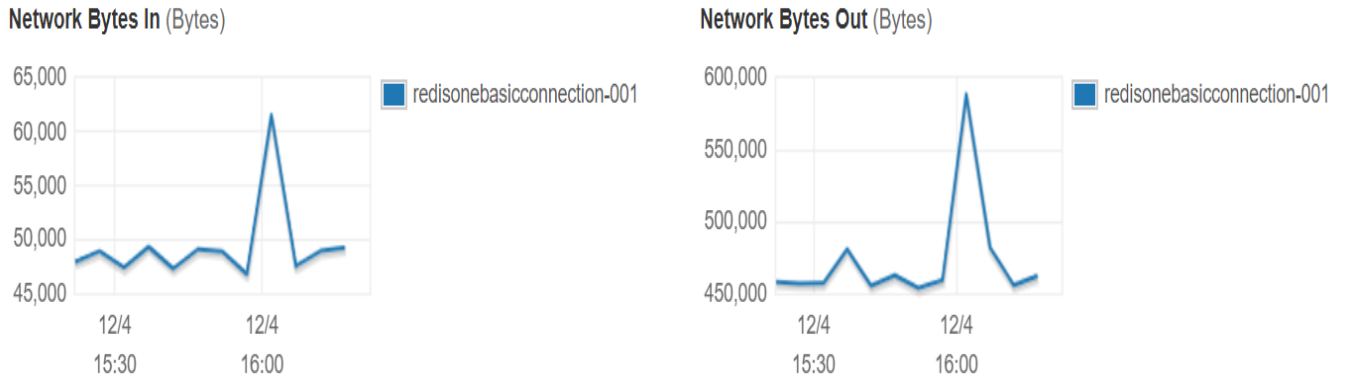


Figure 16: Redis network bytes in and out

Platforms	Time	cost
MARLA Algorithm	18658 ms	0.0000266\$
AWS EMR	28 seconds	0.00086\$
Hadoop MapReduce	183 seconds	-\$
Composite Architecture	12753 ms	0.000555

The table 6.2 shows the different platforms for executing MapReduce in serverless infrastructure. The composite architecture shows 0.5x execution time and 3x cost in compare with Marla model. The cost is due to the usage of cache resources.

As comparison with the variation of execution with different size of files, the composite model gives the efficient time of execution. Below diagram shows the time v/s file size. The file size from 1MB to 5MB are executed in Marla model and composite model. The marla model execution time increases as file size increases. The composite model exhibits the same execution time for all size of data. This is shown in figure [17].

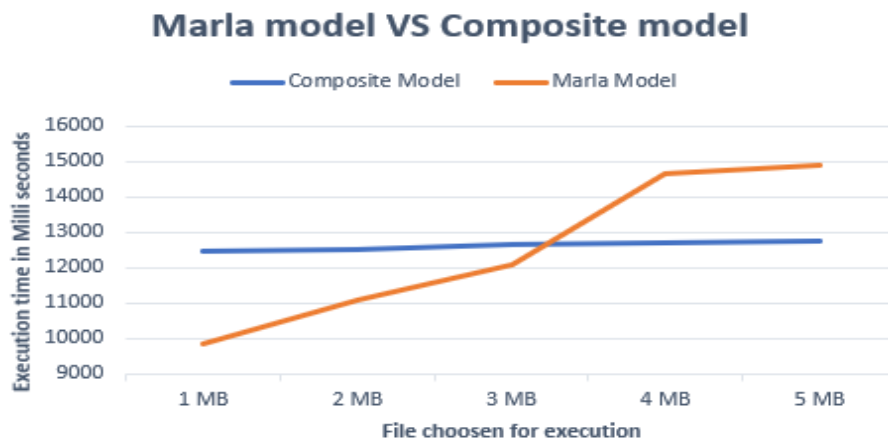


Figure 17: MARLA v/s Composite

The composite model exhibit the greater performance and cost of using cache memory is high. It will be in 3 times more than Marla model. The use of cache memory spin up the large instance for every execution.

## 7 Conclusion and Future Work

This paper has introduced the Composite model in order to support MapReduce in AWS Serverless infrastructure. This paper believes that performance gap is improved by introducing the fine grained elasticity with favour to state of art Function as a service and Backend as a Service. This paper made efficient usage of AWS S3 and AWS Elastic Cache for improved data transfers. When comparing the performance result with other platforms, the AWS Lambda exhibits homogeneous latency behaviour which has no influence on processing time of MapReduce tasks. To this aim, this research take up the deep analysis of AWS Elastic Cache CPU utilization and AWS Lmabda with duration, error rate, invocation and throttle time and pay particular attention to identifying the reasons for latency and billing.

- The introduction of AWS Elastic Cache as intermediate storage reduces the execution time up to 5 times less than the Marla model.
- The usage of cache memory increases cost which is 3 times more than the Marla model.

Due to restriction of resources, this paper proposes the Composite model for small analytic platform. Future work comprise expanding the Composite model for large analytical platform. In particular, creating clusters of lambda and elastic cache in AWS EC2 for executing large data sets. Also, different types of storage's is used for comparing the results and finding more suitability for the Composite model.

## References

- Ahmad, N. M., Yaacob, A. H., Amin, A. H. M. and Kannan, S. (2014). Performance analysis of mapreduce on openstack-based hadoop virtual cluster, *2014 IEEE 2nd International Symposium on Telecommunication Technologies (ISTT)*, IEEE, pp. 132–137.
- Alipour, H., Liu, Y., Hamou-Lhadj, A. and Gorton, I. (2016). Model driven performance simulation of cloud provisioned hadoop mapreduce applications, *Proceedings of the 8th International Workshop on Modeling in Software Engineering*, ACM, pp. 48–54.
- Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A. and Suter, P. (2017). *Serverless Computing: Current Trends and Open Problems*, pp. 1–20.
- Bardsley, D., Ryan, L. and Howard, J. (2018a). Serverless performance and optimization strategies, pp. 19–26.
- Bardsley, D., Ryan, L. and Howard, J. (2018b). Serverless performance and optimization strategies, *2018 IEEE International Conference on Smart Cloud (SmartCloud)*, IEEE, pp. 19–26.
- Cheng, D., Zhou, X., Xu, Y., Liu, L. and Jiang, C. (2018). Deadline-aware mapreduce job scheduling with dynamic resource availability, *IEEE Transactions on Parallel and Distributed Systems* **30**(4): 814–826.
- Clemente-Castelló, F. J., Nicolae, B., Mayo, R. and Fernández, J. C. (2018). Performance model of mapreduce iterative applications for hybrid cloud bursting, *IEEE Transactions on Parallel and Distributed Systems* **29**(8): 1794–1807.
- Cui, Q. and Wu, S. (2018). Astronomical data application research based on mapreduce, *2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, IEEE, pp. 2345–2348.
- Daikoku, H., Kawashima, H. and Tatebe, O. (2018). Skew-aware collective communication for mapreduce shuffling, *2018 IEEE International Conference on Big Data (Big Data)*, IEEE, pp. 3331–3340.
- Dartois, J.-E., Ribeiro, H., Boukhobza, J. and Barais, O. (2019). Cuckoo: Opportunistic mapreduce on ephemeral and heterogeneous cloud resources, pp. 396–403.
- Dashbird (2018). Can serverless computing reshape big data and data science?  
**URL:** <https://dashbird.io/blog/serverless-computing-reshape-big-data-data-science/>
- Elgamal, T. (2018). Costless: Optimizing cost of serverless computing through function fusion and placement, *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, IEEE, pp. 300–312.
- Engdahl, S. (2008). Blogs.  
**URL:** <https://aws.amazon.com/blogs/compute/ad-hoc-big-data-processing-made-simple-with-serverless-mapreduce/>

- Giménez-Alventosa, V., Moltó, G. and Caballer, M. (2019). A framework and a performance assessment for serverless mapreduce on aws lambda, *Future Generation Computer Systems* **97**.
- Goddard, W. (2018). The evolution of cloud computing – where’s it going next?  
**URL:** <https://www.itchronicles.com/cloud/the-evolution-of-cloud-computing-wheres-it-going-next/>
- Guo, J. and Agrawal, G. (2018). Achieving performance and programmability for mapreduce(-like) frameworks, *2018 IEEE 25th International Conference on High Performance Computing (HiPC)* pp. 314–323.
- Hsaini, S., Azzouzi, S. and Charaf, M. E. H. (2018). Testing rules for mapreduce frameworks, pp. 94–98.
- Huang, Y., Yesha, Y., Halem, M., Yesha, Y. and Zhou, S. (2016). Yinmem: A distributed parallel indexed in-memory computation system for large scale data analytics, pp. 214–222.
- Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C.-C., Khandelwal, A., Pu, Q., Shankar, V., Carreira, J., Krauth, K., Yadwadkar, N. et al. (2019). Cloud programming simplified: a berkeley view on serverless computing, *arXiv preprint arXiv:1902.03383*.
- Kaniwa, F., Dinakenyane, O. and Madhav, V. (2017). Parallel algorithm for indexing large dna sequences using mapreduce on hadoop, pp. 1576–1582.
- Kim, Y. and Lin, J. (2018). Serverless data analytics with flint, *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, IEEE, pp. 451–455.
- Klimovic, A., Wang, Y., Kozyrakis, C., Stuedi, P., Pfefferle, J. and Trivedi, A. (2018). Understanding ephemeral storage for serverless analytics, *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, USENIX Association, Boston, MA, pp. 789–794.  
**URL:** <https://www.usenix.org/conference/atc18/presentation/klimovic-serverless>
- Kumar, A., Shankar, R., Choudhary, A. and Thakur, L. S. (2016). A big data mapreduce framework for fault diagnosis in cloud-based manufacturing, *International Journal of Production Research* **54**(23): 7060–7073.
- Li, J., Lin, X., Cui, X. and Ye, Y. (2013). Improving the shuffle of hadoop mapreduce, *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, Vol. 1, pp. 266–273.
- Matthews, S. and Leger, A. (2017). Leveraging mapreduce and synchrophasors for real-time anomaly detection in the smart grid, *IEEE Transactions on Emerging Topics in Computing* **PP**: 1–1.
- McGrath, G. and Brenner, P. R. (2017). Serverless computing: Design, implementation, and performance, *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp. 405–410.
- Moczurad, P. and Malawski, M. (2018). Visual-textual framework for serverless computation: A luna language approach, *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pp. 169–174.

- Nabavinejad, S. M. and Goudarzi, M. (2017). Faster mapreduce computation on clouds through better performance estimation, *IEEE Transactions on Cloud Computing* .
- Namboori, R. (2019). The evolution of cloud computing - dzone cloud.  
**URL:** <https://dzone.com/articles/cloud-computing-1>
- Nicolae, B., Costa, C. H. A., Misale, C., Katrinis, K. and Park, Y. (2017). Leveraging adaptive i/o to optimize collective data shuffling patterns for big data analytics, *IEEE Transactions on Parallel and Distributed Systems* **28**(6): 1663–1674.
- Pelle, I., Czentye, J., Dóka, J. and Sonkoly, B. (2019). Towards latency sensitive cloud native applications: A performance study on aws, *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 272–280.
- Pu, Q., Venkataraman, S. and Stoica, I. (2019). Shuffling, fast and slow: Scalable analytics on serverless infrastructure, *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, USENIX Association, Boston, MA, pp. 193–206.  
**URL:** <https://www.usenix.org/conference/nsdi19/presentation/pu>
- Raj, E. D. and Babu, L. D. (2015). A two pass scheduling policy based resource allocation for mapreduce, *Procedia Computer Science* **46**: 627–634.
- Samadi, Y., Zbakh, M. and Tadonki, C. (2018). Analyzing fault tolerance mechanism of hadoop mapreduce under different type of failures, pp. 1–7.
- SeasiaInfotechBlog (2019). History evolution of cloud computing : What to expect in 2019.  
**URL:** <https://www.seasiainfotech.com/blog/history-and-evolution-cloud-computing/>
- Sharma, A. K. and Singh, G. (2018). A review on data locality in hadoop mapreduce, *2018 Fifth International Conference on Parallel, Distributed and Grid Computing (PDGC)* pp. 723–728.
- Wasi-ur Rahman, M., Islam, N. S., Lu, X., Jose, J., Subramoni, H., Wang, H. and Panda, D. K. D. (2013). High-performance rdma-based design of hadoop mapreduce over infiniband, *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, IEEE, pp. 1908–1917.
- Wei Zhang, Baolu Li and Kun Li (2013). Multi-pattern matching algorithm based on mapreduce and hadoop, *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)*, pp. 1856–1859.
- Werner, S., Kuhlenkamp, J., Klems, M., Muller, J. and Tai, S. (2018). Serverless big data processing using matrix multiplication as example, *2018 IEEE International Conference on Big Data (Big Data)* pp. 358–365.
- Xenonstack (2018). Cloud transformation, enterprise data and ai platform.  
**URL:** <https://www.xenonstack.com/>
- Yu, W., Wang, Y., Que, X. and Xu, C. (2015). Virtual shuffling for efficient data movement in mapreduce, *IEEE Transactions on Computers* **64**(2): 556–568.



# Configuration Manual

MSc Research Project  
Cloud Computing

Achyut Anantakumar Vadavadagi

Student ID: x18131557

School of Computing  
National College of Ireland

Supervisor: Manuel Tova-Izquiere

National College of Ireland  
Project Submission Sheet  
School of Computing



<b>Student Name:</b>	Achyut Anantakumar Vadavadagi
<b>Student ID:</b>	x18131557
<b>Programme:</b>	Cloud Computing
<b>Year:</b>	2019
<b>Module:</b>	MSc Research Project
<b>Supervisor:</b>	Manuel Tova-Izquredo
<b>Submission Due Date:</b>	11/12/2019
<b>Project Title:</b>	Configuration Manual
<b>Word Count:</b>	1000
<b>Page Count:</b>	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

**ALL** internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

I agree to an electronic copy of my thesis being made publicly available on TRAP the National College of Ireland's Institutional Repository for consultation.

<b>Signature:</b>	
<b>Date:</b>	10th December 2019

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:**

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
<b>Attach a Moodle submission receipt of the online project submission</b> , to each project (including multiple copies).	<input type="checkbox"/>
<b>You must ensure that you retain a HARD COPY of the project</b> , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

<b>Office Use Only</b>	
Signature:	
Date:	
Penalty Applied (if applicable):	

# Configuration Manual

Achyut Anantakumar Vadavadagi  
x18131557

## 1 Introduction

### 1.1 Purpose of this document

This Configuration Manual is done based on the NCI Research project requirements described in the Project Handbook. Main purpose of this document is to describe the required software tools and settings in order to successfully deploy the Spot instance Management System.

### 1.2 Document Structure

Section	Purpose
General Information	This module explains the minimum information and prerequisites required for the serverless platform setup
Development Environment prerequisites	This module explains steps required for successful setup of development environment used for development and update of the solution Solution
Solution Deployment Procedure	This module explains deployment procedure of the Composite model
Validations	This module explains the minimum requirements to validate fruitful deployment of the solution

## 2 General Information

### 2.1 Objective

The main objective of Composite model is to enhance the shuffling phase in Marla model which is defined in serverless infrastructure. The key role is to run the MapReduce tasks using AWS services and adding fine grained elasticity. Analysing the cost and speed of the different type of storage's and other platforms to provide the best fit for the MapReduce tasks.

### 2.2 Solution Summary

The Composite model solution consists of six phases which are interconnected and acts as one architecture.

- The input phase takes comma separated files which is added into the s3 bucket and acts as the event to trigger the execution of MapReduce tasks.
- The coordinator phase takes input data and divides into data pieces at specified size and guarantees that all the given input data is executed in map lambda function. This stage also helps to trigger the map function where the function is defined.
- In third phase, mapper lambda function divides the given input data into required and non-required data where it is used for the further execution. This function returns the key value pairs and stored in to the elastic cache memory.
- In the shuffle phase the data from all map functions is collected and stored in the AWS Redis storage. The data is key value pair called as intermediate data.
- This fifth phase the reduce function takes the intermediate data from the Redis and returns the two dimensional key value pairs.
- In sixth phase the output from the reduce function output is stored in to the AWS S3 in text file.

## **2.3 Architecture Requirements**

This section describes the required AWS services for building Composite model.

### **2.3.1 Amazon Web Services account**

The AWS serverless platform account is created with the user details (Vichi (2015)).

### **2.3.2 AWS Lambda**

The Lambda compute service provided by the Amazon is required for the creating coordinator, map and reduce functions. Here functions are written in python 3.8 is used for executing MapReduce tasks (Hendrix (1983)).

### **2.3.3 AWS S3**

The AWS S3 (simple storage service) is used for storing input and output files. It is also used for triggering the execution of MapReduce tasks (Street (2002)).

### **2.3.4 AWS VPC**

The AWS VPC (Virtual Private Network) is created to provide the internet access to lambda and elasti cache services. Here subnets, security groups, route table and NAT gateway to support VPC (de Médicos de (2016)).

### **2.3.5 AWS Cloud Watch**

The AWS Cloud Watch service is used for analysing monitoring, error handling, analysing logs and so on. This service provides all the information required for executing Mapreduce tasks at a given time (Jenkins (2000)).

### **2.3.6 AWS EC2**

The AWS EC2 service provided by the Amazon is used for the creating libraries which are dependent for executing the tasks (Amazon (2018a)).

### 2.3.7 AWS Elastic Cache

It is used for storing the intermediate files that are produced from map functions. This is mainly used as fine grained elasticity for composite model (Amazon (2018b)).

## 2.4 Required Skills

In this guide, we assume that user has basic knowledge of using Amazon Web Services. Also user has to know the python language for creating all functions.

# 3 Development Environment Requirements

## 3.1 Code Repository

Refer the zip file which I have submitted in the ICT solution.

## 3.2 Required Programming Languages

Python Version 3.8

The AWS compute engine provides the platform to write the python code for creating MapReduce functions. The following packages need to be installed before writing MapReduce functions:

- boto3 - 1.10.28
- redis - 5.0
- pandas - 0.24.2
- csv -13.1.1

## 3.3 Creating IAM role

Before creating Lambda function, create the IAM role which has below permissions. This sets the lambda to create the execution environment.

- AmazonElasticCacheFullAccess
- AmazonS3FullAccess
- CloudWatchFullAccess
- AmazonVPCFullAccess
- AWSLambdaVPCAccessExecutionRole
- AWSLambdaRole
- CloudWatchEventsFullAccess

### 3.4 Create Lambda function

Once the IAM role is created, you can create the lambda functions for coordinator, map and reduce function. This is shown below:

- Click on create function by selecting the lambda service as shown in figure 1.
- Select the blueprint after clicking the create lambda function as shown in figure 2.
- Configure the function name, run-time and permissions as shown in figure 3]
- After creating the lambda function configure the network as shown in figure 5
- Add the event that triggers the created lambda function. If there is no S3 bucket created, create the S3 and add it to the lambda function

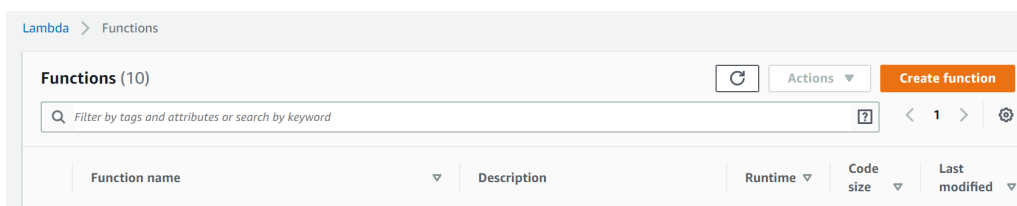


Figure 1: Creating the function

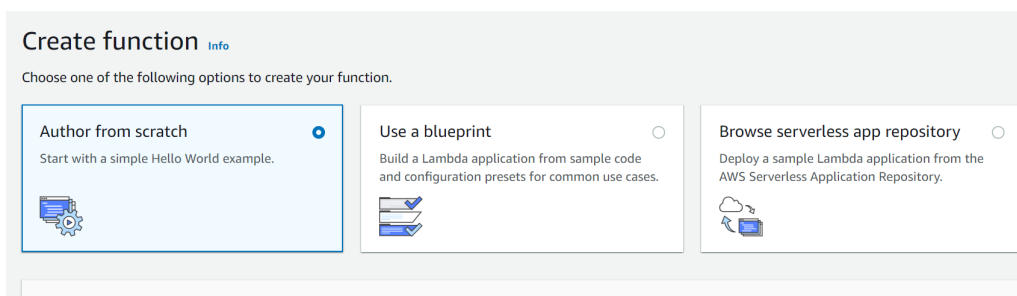


Figure 2: Select blueprint

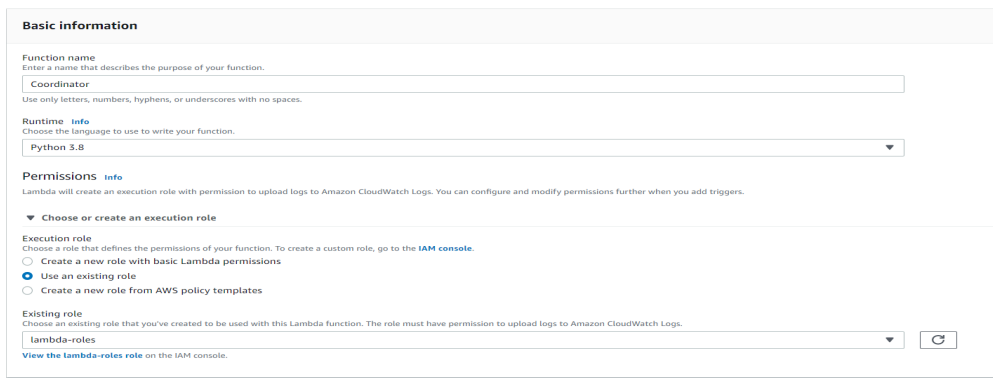


Figure 3: Configuration image

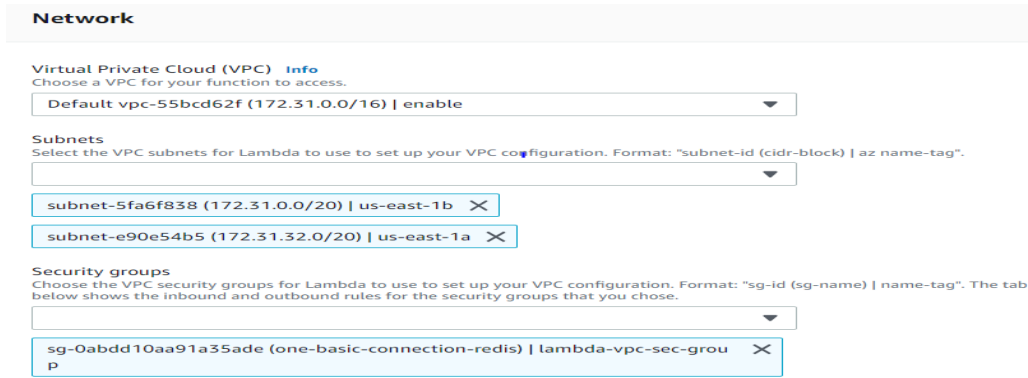


Figure 4: Network Configuration



Figure 5: Triggering event S3

### 3.5 Create the EC2 instance

After creating the lambda function with above functionalities, create the EC2 cluster for adding the dependencies files to the lambda function.

- Click on create function by selecting the Ec2 service as shown in figure 6.
- Install the Filezilla to transfer the local python file to EC2. After installing, add the EC2 public IP and with port number 22 as shown in figure 7.
- Transfer the local file to EC2 as shown in figure 8
- Later install all the dependencies inside the EC2 instance
- Move the installed files to S3 buckets which you have been created

After adding the libraries to S3 buckets, create the layer inside the lambda function to add dependencies to the MapReduce tasks as shown in figure 9.

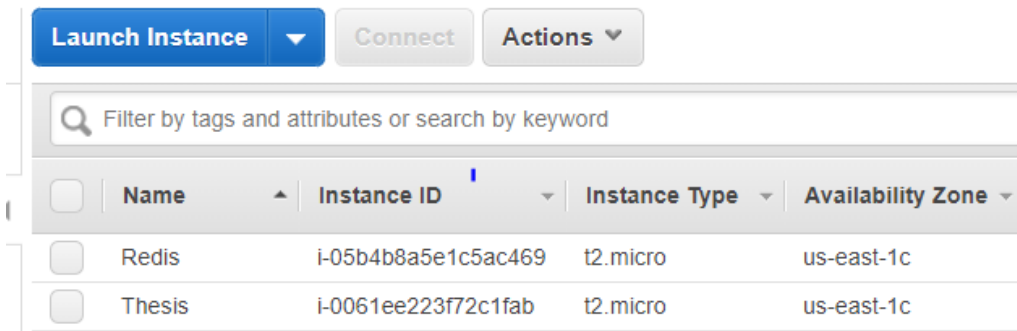


Figure 6: Creating EC2 instance

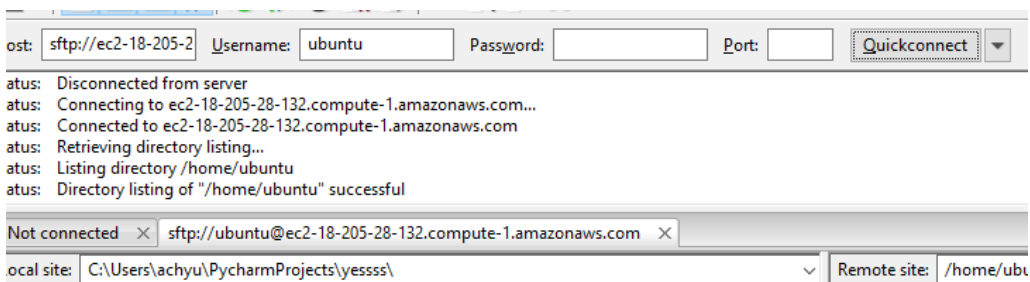


Figure 7: File Zilla Configuration

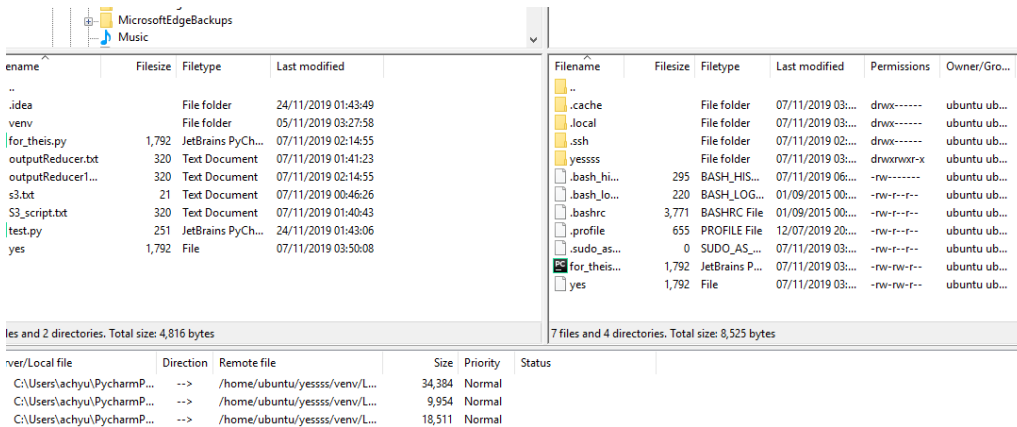


Figure 8: Adding files to file zilla

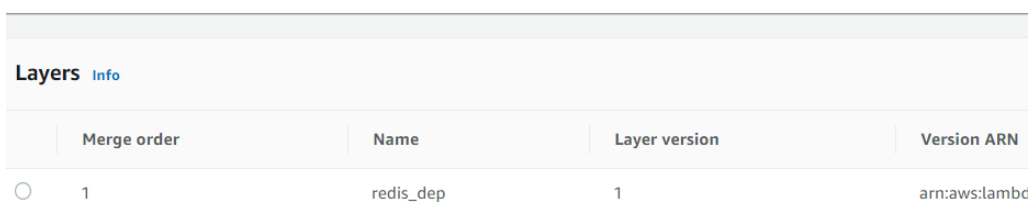


Figure 9: Layer Structure



### 3.6 Create Cache Memory

Create the cache memory by selecting the Redis inside AWS Elastic Cache in services. Create the Redis service with configuration as shown in figure 10.



Figure 10: Redis Configuration

## 4 Validation

Create the lambda function for Coordinator, mapper and reducer function. The code is available in section 2 and deploy it inside lambda function. After creation of all services shown, test the configurations and application from coordinator lambda function. There are two types of validations are done as shown below.

- Create the test inside the lambda function as shown in figure 11. Execute the test function and result is shown in figure 12. If there is no proper configuration made, an error is thrown.
- Add the files inside the S3 buckets as shown in figure 13. It will execute the lambda function automatically. If there is proper connection made, the output is displayed in cloud watch service as shown in figure 14.

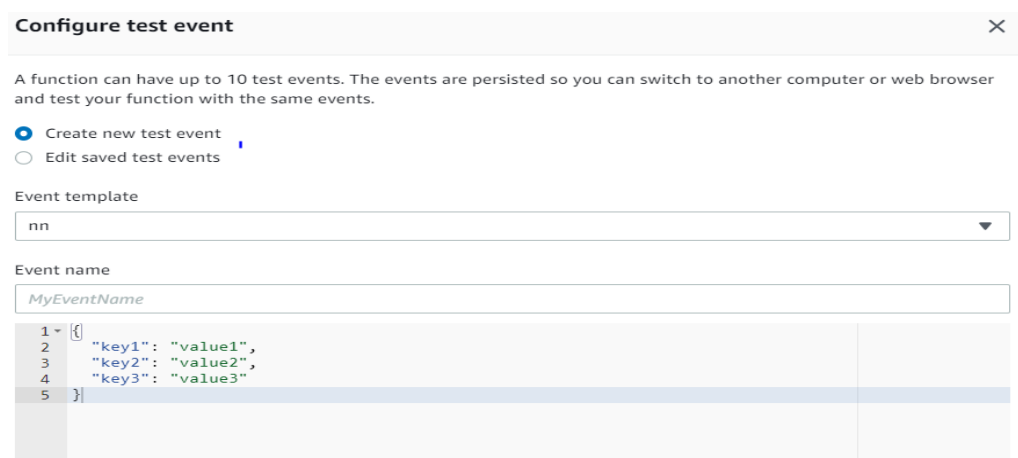


Figure 11: Creating test event

Code SHA-256 MIPnn+RJKdUC2xROGkym7Trufo1CDNY4TCd15Ru+ydU=	Request ID 74ef534e-ca91-4d90-941b-ca162c140498
Duration 12638.28 ms	Billed duration 12700 ms
Resources configured 704 MB	Max memory used 131 MB

**Log output**  
The section below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. [Click here](#) to view the CloudWatch log group.

```

START RequestId: 74ef534e-ca91-4d90-941b-ca162c140498 Version: $LATEST
FileSize = 2.9184627532958984 MB
chunk size to small (1 bytes), changing to 1024 bytes
numberMappers-- -1021.0815372467041
chunk size to large (1024 bytes), changing to 460 bytes
END RequestId: 74ef534e-ca91-4d90-941b-ca162c140498
REPORT RequestId: 74ef534e-ca91-4d90-941b-ca162c140498 Duration: 12638.28 ms Billed Duration: 12700 ms Memory Size: 704 MB Max Memory Used: 131 MB

```

Figure 12: Logs from test in lambda

onebasicconnectionextend

Overview Properties Permissions Management Access points

🔍 Type a prefix and press Enter to search. Press ESC to clear.

📤 Upload 📁 Create folder 📄 Download ⌵ Actions

<input type="checkbox"/>	Name	Last modified
<input type="checkbox"/>	📁 tmp	--
<input type="checkbox"/>	📄 SFIGenderDashboard_TableauPublic_2019 - Copy.csv	Dec 4, 2019 11:45:13
<input type="checkbox"/>	📄 SFIGenderDashboard_TableauPublic_2019.csv	Dec 2, 2019 5:47:51
<input type="checkbox"/>	📄 redis.zip	Nov 9, 2019 11:49:14

Figure 13: S3 as trigger function

```

▶ 11:50:31 chunk size to small (2 bytes), changing to 1024 bytes
▶ 11:50:31 numberMappers-- -1018.7937278747559
▶ 11:50:31 chunk size to large (1024 bytes), changing to 460 bytes
▶ 11:50:43 END RequestId: 72c73c22-01c9-4d5c-8722-4c8b4e5a233d
▶ 11:50:43 REPORT RequestId: 72c73c22-01c9-4d5c-8722-4c8b4e5a233d Duration: 12917.55 ms Billed Duration: 13000 ms Memory Size: 704 MB Max Memory Used:
▶ 11:51:11 START RequestId: 98476717-de5f-4928-9263-e217da2c109d Version: $LATEST
▶ 11:51:11 FileSize = 5.206272125244141 MB

```

Figure 14: Logs on Cloud Watch

## References

Amazon (2018a). Clouds project ec2.

**URL:** <https://aws.amazon.com/ec2/>

Amazon (2018b). Clouds project elastic cache.

**URL:** <https://aws.amazon.com/elasticache/>

de Médicos de, C. (2016). Vpc: Validación periódica de la colegiación.

**URL:** <https://aws.amazon.com/vpc/>

Hendrix, R. W. (1983). Lambda.

**URL:** <https://aws.amazon.com/lambda/>

Jenkins, G. (2000). Clouds project cloudwatch.

**URL:** <https://aws.amazon.com/cloudwatch/>

Street, S. (2002). S3.

**URL:** <https://aws.amazon.com/s3/>

Vichi, M. (2015). Il console.

**URL:** <https://aws.amazon.com/console/>