National College of Ireland

Enhanced genetic algorithm to reduce makespan of multiple jobs in map-reduce application on serverless platform

MSc Research Project
MSc in cloud computing

Divya B. Thorat
Student ID: X18191878

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Divya Thorat
Student ID:	x18191878
Programme:	Masters in cloud computing
Year:	2020
Module:	MSc Research Project
Supervisor:	vikas sahni
Submission Due Date:	17/08/2020
Project Title:	Enhanced genetic algorithm to reduce makespan of multiple jobs in map-reduce application on serverless platform
Word Count:	6180
Page Count:	20

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.

Signature:	Divya B. Thorat
Date:	23rd September 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

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

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

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Enhanced genetic algorithm to reduce makespan of multiple jobs in map-reduce application on serverless platform

Divya Thorat
x18191878

Abstract

Nowadays, allocating proper tasks to the resources is an integral part of the cloud environment. So the execution time is depending on the number of resources allocated to the environment. So it necessary to choose the proper scheduling algorithm for multiple applications. The serverless platform is the combination of function as a service and back-end as service. This paper proposed a map-reduce jobs with a genetic algorithm on a serverless platform. In this, we have used Lambda function as a service and s3 bucket, Redis storage as back end as service. The combination of fast and slow storage gives the fine-grained elasticity, and a genetic algorithm minimizes the total execution time. In a serverless platform, The pricing depends upon the number of times the function executed and the total execution time required for operation. The genetic algorithm requires low execution time, so it reduces the cost of the operation, and a combination of slow-fast storage gives better performance along with efficiency. The evaluation carried out on a serverless platform, comparison carried out between map-reduce application without genetic algorithm and with a genetic algorithm, so result shows the map-reduce application with a genetic algorithm requires less execution time and has higher performance.

1 Introduction

Nowadays, a data operation centers and storage operating centers are in more numbers as compared to past days, and the data generated through these operation centers are in huge numbers. It is challenging to manage massive data, so it is necessary to adapt the management of resources and tasks in a productive manner. It enhances the efficiency and performance of the operations. Management of the task is challenging in all the cloud service model. In software as service and platform as service models, management of resources and task scheduling are the most crucial factor as it manages how much amount of resources needs to allocate for a particular job Buyya et al. (2018).As currently in a data center, most of the applications are running in parallel phase, so the utilization of resources is decreased, so resource allocation must be in a proper manner to improve the performance and efficiency of the system. So task scheduling plays a important part in assigning the task to applications Arunarani et al. (2019). Serverless domain for cloud operator has the challenge such as the appropriate allocation of resources for each service must be used to increase the total income while maintaining the user satisfaction for the service along with service as function and service quality. So in function as a

service domain, this problem is taken into consideration, its solution is to minimize the total time of implementation of the jobs and efficient provisioning of the resources, so it reduces the latency issue. Minimizing the execution time of application requires to choose the appropriate scheduling algorithm Buyya et al. (2018). Recently map-reduce application plays a vital role in big data processing to process the massive amount of data. Massive data processing with map-reduce applications requires high expenses. However, if we implement the map-reduce jobs on a serverless platform, then there is no need to maintain the servers, and the serverless platform is cheaper as compared to MapReduce application on the Hadoop platform [1].

In a task scheduling mechanism, the efficiency and system performance increased by allocating proper tasks to processors, so it reduces the execution time Akbari et al. (2017). The primary goal of task management is a load balancing, scalability, reliability, and dynamic allocation of the resources so that it reduces the problems of task scheduling Arunarani et al. (2019).

The static algorithm for scheduling classifies in heuristic and meta-heuristic methods. The heuristic approach is classified in lists, clusters and duplication methods. The heuristic process could not produce permanent results for several problems as time complexity of the scheduling increases; also, this method increases the computing cost.A meta-heuristic method provides a high level of ergodicity with a flexible and straightforward architecture; also, it gives the solution for optimization issues Akbari et al. (2017). The meta-heuristic method further categorizes into different methods.

Map-reduce application with the serverless platform processing the data without any operational hassle, and there is no need to have the Hadoop knowledge. It reduces the operational cost as there is no need to manage the servers, and the user has to pay only for execution of function. In a serverless platform, operational cost depends on how many times the function has been executed, so if running time is long, the operating cost is also very high. The genetic algorithm's execution time is low as compared to the other algorithm; it best suited for MapReduce jobs on a serverless platform [2]. many of the researchers have chosen genetic algorithm as it has low execution time Arunarani et al. (2019). This algorithm used to minimize the run-time and provides the optimal solution. The genetic algorithm uses the operator, such as a selection, crossover, and mutation, for getting the operation end requirement fulfilled Sathya Sofia et al. (2019).

when using the map-reduce application, It is important to have the operations with excellent performance while maintaining the application cost-efficient. So In the proposed architecture, we have used the slow storage for cost reduction, fast storage for getting high performance, and a genetic algorithm to minimizes the total completion time, so total operation cost reduced along with better efficiency Pu (Feb 2019).

1.1 Research Question:

"Can execution time of multiple jobs in map-reduce applications be reduce using enhanced genetic algorithm as it requires less execution time and gives more correctness in the solution?"

[1] url:https://aws.amazon.com/blogs/compute/ ad-hoc-big-data-processing-made-simple-with-serverless-mapreduce/

[2] url: https://aws.amazon.com/blogs/compute/ ad-hoc-big-data-processing-made-simple-with-serverless-mapreduce/

2 Related Work

In this area we have performed a survey for choosing a scheduling algorithm for map-reduce application, which platform is suitable to executing the application and how we can achieve high performance along with reduction in cost of the operation. The subsection 2.1 represent the overview on task management in cloud computing ,subsection 2.2 represent the review on task scheduling with genetic algorithm, 2.3overview on map-reduce jobs on serverless platform and subsection 2.4 represent Literature review analysis.

2.1 Overview of Task scheduling in Cloud computing

Nowadays, parallelism increased, so the utilization of the resources reduced, so we have to do the job scheduling appropriately to improve the performance. Arunarani et al. (2019) paper makes research on job scheduling strategies and represents the performance of cloud computing. The objective of the task management is scalability, task scheduling, reliability, and dynamic resource allocation to nodes. Scheduling strategies classified as Qos-based, Ant colony optimization, PSO based,fuzzy-based, and cost-based. To analyze the performance of each scheduling type researcher refer the six different paper and concluded that the Genetic algorithm(GA) and particle swarm optimization is the best among them. Asghari and Navimipour (2018)show different types of task scheduling algorithms are available, its pros and cons; with the help of such research, The algorithm most suitable can be found easily.

Cloud computing has an issue like taking the correct decision for assigning the task to the resources is difficult, and it affects the completion time of the system, so the use need to pay further cost for the same. To avoid such circumstances, one needs to assign a proper task scheduling method for data-intensive and big data applications. most of the independent tasks scheduled with the basic genetic algorithm. Although many applications are complicated, they have subtasks to communicate internally. Gawanmeh et al. (2018)have presented a paper where it can schedule the dependent task following the user's requirement with precise time and price. The proposed method gives excellent performance than the standard algorithm. It is restricted to one vector and one vector.

To execute the task with low energy consumption is the main objective of a user in cloud computing, so the cost of the operation reduced. Sathya Sofia et al. (2019) proposed a non-dominated dual-target sorting algorithm. To perform the dual objective, they considered ten, fifteen tasks and subtask respectively for fifteen heterogeneous resources. The design function as a goal and the design limitations are another function. For cloud users, the total time of execution and operating costs are two key factors and we have to take these as several design goals. The research method presented calculated the objective role and the distance between the crowds. After performing the standard operation, parent is selected using tournament method. The numerical results show total time of performance and operating costs achieved using the multi-target optimisation methods.

2.2 Overview of task scheduling with genetic algorithm

As the resource competition increased in cloud computing, map-reduce jobs run without any input data, so it may impact the processing of the data and causes the delay. Map-reduce applications cluster efficiency decreases due to data locality. The best way to improve the data location of the map application is to cache the input data into the

memory. But it is hard to decide when and what to pick up in cluster design. So Sun et al. (2016) has Presented the scheduling planner to improve the data location. In the presented paper, it finds outs the pending jobs, and then in the future, it Predicts the task node so that it can preload the memory.

In a computation grid, it is necessary to utilize the resources properly, so it has to use the appropriate scheduling methods. Rajeswari et al. (2019) proposed map-reducer model with an algorithm of the non-dominated sorting in computation grid. They considered the overall run time and flow time to minimize the use of resources. The author compares NSGA-II with a "weighted multi-target genetic algorithm" using a bench marking tool, which states that genetic algorithm is low in execution time in comparison to other algorithms. Zhang et al. (2019)The researcher also uses the NSGA -II algorithm, but for the initial population, it uses the heuristic two-stage allocation method and finds the best solution. Jena and Mohanty (2018) used the genetic algorithm based on user conscious task scheduling.In heterogeneous environments,Akbari et al. (2017) used a modified version of genetic algorithm for task planning. Zhang et al. (2019),Costa et al. (2020) and Luo et al. (2020) use a genetic algorithm for job shop task assigning methods. Enes et al. (2020) show what is the need of a serverless environment, it's problems, and how it can solve these problems.

2.3 Review on map-reduce application on serverless platform

Nowadays, in a serverless computing application, resource provisioning and scaling are performed by a third-party vendor like AWS lambda, Azure functions, and google cloud function. Wang et al. (2018) have presented a paper where they represent the analysis of the three primary FAAS cloud providers based on resource utilization, performance isolation efficiency, and architecture. So they represent some highlights of the results,

- AWS lambda gives better scalability and lowers cold start latency over google cloud function. However, lambda gives a lack of isolation among the instances that causes a decrease in I/O, networking, and cold start performance.
- Half of the time, Azure functions run on a virtual machine with a loss of performance.
- As the google cloud function has the accounting issue, it allows the user to use function instance to achieve the computing resource small as a virtual machine to keep the cost almost zero.

To analyze the large dataset, mostly the map-reduce programming model is us. Nowadays, function as service primarily used as there is no need to manage the servers. Serverless computing allocates the resources dynamically, and pricing based on the execution time. So, Giménez-Alventosa et al. (2019) have introduced the serverless architecture for running the map-reduce application on AWS lambda using Amazon s3. In this, it introduces the MARLA framework to execute the python-based map-reduce jobs on Aws lambda service. This framework depends on the AWS services as when the dataset uploaded to s3 service, and it invokes the coordinator lambda function. By dividing the total size into many chunks coordinator function calculates the size of partition and allocated these chunks to mappers. If the chunk size is not satisfied, then the coordinator function checks some conditions such as if the chunk size is smallest than the minimum block, if the chunk size larger than the safe memory size, and if the chunk size is bigger

than the maximum block size. In this, it also adds an extra mapper to process the residual data. The mapping process produces a list of key/value pairs and these chunks assigned to independent reducer lambda function. This MARLA architecture handles the failure at three stages that are a failure on the coordinator, failure on the mapper function, and failure on the reduce function.

The fine-grained elasticity of the serverless computing is useful for high resource utilization in an application like analytics workload. However, the resource limits make it challenging as they have to move a large amount of data between two functions. So, Pu (Feb 2019)have presented the paper where they use the locus serverless analytics platform. That combines slow storage, i.e., amazon s3, DynamoDB, and fast storage, i.e., elastic cache, radios. This combination gives excellent performance with cost efficiency. By using this , customer can select the appropriate storage mechanism and parallelism for applications such as map-reduce. Locus reduces the total cluster time by 59% represented in total core-seconds, which is nearby to apache spark's completion time by up to twice. Locus equivalent to Apache spark in running time measured on cloud sort benchmark even though it is using a small amount of fast storage. This paper concluded that Locus is twice slower than the amazon redshift, but still, it is a good decision as it doesn't require any provisioning time. By expanding the pywren (data analytic engine) Locus designed. This paper uses S3 as slow storage and Redis nodes on Amazone elasticache.

Malawski et al. (2020)Has presented a paper where they have implemented a prototype execution workflow using AWS lambda, google cloud functions, and hyper flow workflow engine. In this, they have mentioned what the main feature of AWS lambda is, google cloud function and azure. Also, they showed what the maximum limit for execution time, disk space, several functions, and parallel execution is. The prototype execution workflow evaluated with montage application.

Liroz-Gistau et al. (2016)Has presented a paper where they have introduced intermediate reduce phase to solve the reduce data skew problem and process the data parallelly more efficiently.

Marra et al. (2020)Has presented a paper where they have shown how the debugging approach work for big data application such as MapReduce application.

Ruiz et al. (2016)Has presented paper a where they have used the formal model to improve the performance and cost agreement in Hadoop environment. The process of model is implemented using Timed process algebra BTC.

2.4 Analysis of Literature review:

In Table 1 shows the analysis of the research papers. In this it shows the analysis of, on which area the research should be , what is the necessity of task scheduling and selection of task scheduling algorithm, Selection among the three FAAS cloud provider, mapreduce job on serverless platform and better performance with low latency.

Selection of criteria	**Title of the Papers**	**Author**	**Importance of research**
Area of research	"A framework and a performance assessment for serverless MapReduce on AWS lambda"	Gimenez alventosa	map-reduce application implemented on serverless but it has performance and latency issue.
The necessity of task scheduling and Selection of a task scheduling algorithm	"Task scheduling techniques in cloud computing "	arunarani task 2019	research on job scheduling strategies and represents the performance of cloud computing.
Selection among the three FAAS cloud provider	"scheduling algorithm based on prefetching in MapReduce clusters"	Sun, M(2016)	This paper find out the pending jobs while scheduling the task using a genetic algorithm and predict the future task.
MapReduce jobs on serverless platform	"Peeking behind the curtains of serverless platforms"	Wang, L.(2018)	presented a paper where they represent the analysis of the three primary FAAS cloud providers based on resource utilization, performance isolation efficiency, and architecture
Better performance with low latency	"Shuffling, fast and slow: Scalable analytics on serverless infrastructure"	Pu, Q. (Feb 2019)	presented the paper where they use the locus serverless analytics platform. That combines slow storage, i.e., amazon s3, DynamoDB, and fast storage, i.e., elastic cache, radios. This combination gives excellent performance with cost efficiency.

Table 1: Research Areas with Importance of research

3 Methodology

3.1 Steps:

With lambda, customers can run the code without any specific infrastructure, so that many applications, such as massive data processing are stored on the S3 bucket. Lambda service is the critical element of serverless infrastructure. The project consists of three sections

1. Mapper
2. Coordinator
3. Reducer

In a project following s procedure has been followed to execute the map-reduce application on serverless infrastructure.

1. The s3 bucket designed to store static and dynamic data and the Lambda Coordinator function, created to connect the processing map phase to reduce stage.. To fetch data from s3 to lambda function, we need to give some permission to the IAM role. The IAM role permissions include S3 full access, Cloud watch full access, s3 read-only access, VPC full access, elastic file system full access, lambda role, lambda VPC access executive role, and cloud watch events full access. The S3 trigger event is created to trigger the lambda function when data is inserted into a bucket. In this project, the coordinator function is used to distribute the data to two mapper functions with some conditions according to the data size.
2. Mapper function is invoked from the coordinator function and find out the best solution for the available problem with the help of a genetic algorithm. Each mapper's data is stored into Redis in-memory data storage to provide the excellent grained elasticity.
3. Then the reducer function is invoked, and aggregated output is stored in another s3 bucket. To provide security for the applications, process the data with VPC by configuring the lambda function And creating the VPC endpoint for AWS S3 service.

Figure 1 shows the execution flow for the map-reduce application execution on serverless platform.

3.2 Material and equipment:

3.2.1 AWS services:

IAM role: In a project, a role is an entity that you can create in your account and has dedicated permission. With the help of a role, the user has representative access with predefined permission to the credible entity without sharing long time access [3]. In this project, as we need to access the different services such as s3, AWS lambda, etc., so

[3]url:https://docs.aws.amazon.com/IAM/latest/UserGuide/idroles.html

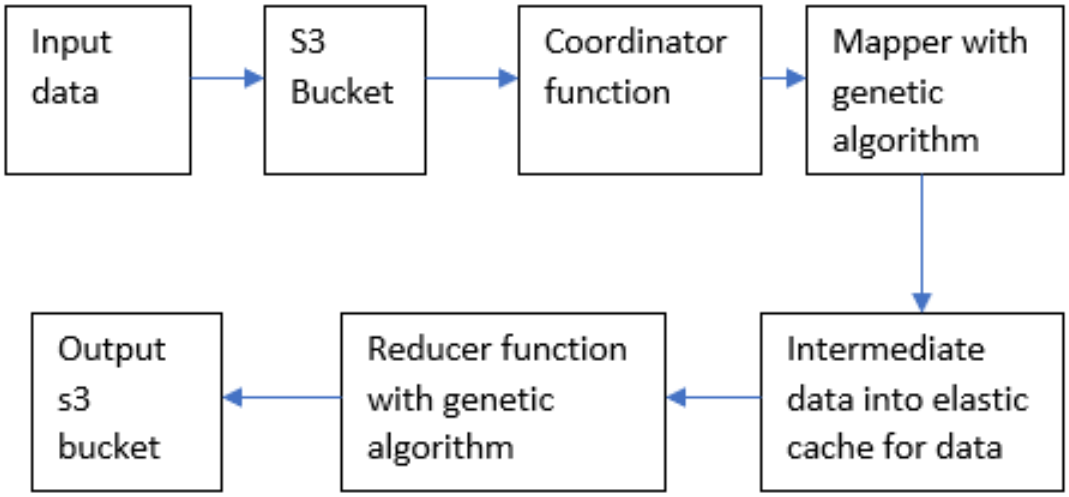

Figure 1: Execution workflow of map-reduce jobs

we have created a role who's name is lambdapermission with access to AmazonElasticCacheFullAccess, AmazonS3FullAccess, CloudWatchFullAccess, AmazonVPCFullAccess, AWSLambdaVPCAccessExecutionRole, AWSLambdaRole, CloudWatchEventsFullAccess and AWSLambdaENIManagementAccess policies .

S3 bucket: it is an object storage service used by customers of all sizes and companies for storing and protecting any amount of data for multiple applications. It gives the 99.99999999999% of durability to store data for multiple application of companies [4]. Two s3 buckets are used in this project to store input and output data. Input buckets name is AWS-lambda-trigger1, and one trigger event is created so that as soon as the data is coming into an Input s3 bucket, it will invoke the coordinator lambda function for execution. Data coming out of the redis function will store in to S3 bucket.

Aws cloud watch logs: it's log is used to monitor, store, and access the log files generated from EC2 instances, s3, Aws Lambda, and many other sources. It also enables users to centralize the logs generated from many sources [5]. In this project, when the lambda function is executed, such as coordinator, mapper, and reducer, its output logs are generated in cloud watch logs.

AWS Lambda: this service enables the user to run the code without provisioning and monitoring the servers. We just have to pay for compute the time of the service. It also provides continuous scaling whenever it will require [6]. In this project, lambda service is used to create the coordinator, mapper, and reducer function. Coordinator function is used to distribute the data across the mapper function, each mapper function is used to find out the best output solution with the help of a genetic algorithm, and reducer function is used to aggregate the output of the mapper function. Lambda function configured with the role, s3 trigger, network configuration , security group, and during run time python 3.8 has been added.

AWS Redis in-memory data storage: Remote dictionary server is a key-value open source storage devices for data storage and database use. [7]. In this project, To

[4]url:https://aws.amazon.com/s3/
[5]url:https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/WhatIsCloudWatchLogs.html
[6]url:https://aws.amazon.com/lambda/
[7]url:https://aws.amazon.com/redis/

provide fast storage with good performance, Redis is used to store the intermediate data generated by the mapper function.

3.2.2 Algorithm Used:

Genetic algorithm: The genetic algorithm is used to schedule tasks since less completion time is necessary. Prices in serverless computing are based on computational time and we have used the genetic algorithm to reduce the total run-time. The genetic algorithm consists of various operator types like selection, crossover, mutation and reverse.

3.2.3 Programming language:

The programming language used is Python 3.8 to create the functions. In this PyGAD is an open-source python 3 libraries for implementing the genetic algorithm. PyGAD provides assistance for distinct types of operators such as mutation, cross over, and parent selection. It allows a distinct type of problem is optimized using the genetic algorithm by the fitness function.

3.2.4 Sample Data:

The data used for mapping are country-based data in the United States. Thus the number of times the country's name in a comma-separated file has been performed. The mapping function produces the key value pair by gathering the data and separates the required and unused data. Then reducer adds the key value pair and gives the country name in the CSV file.

4 Design Specification

Giménez-Alventosa et al. (2019) has proposed a python-based model that can work with MapReduce on serverless platform, with the help of AWS lambda service. In this architecture It stores the input data into s3 bucket , intermediate data generated from mapper and output data generated from reducer are stored into the s3 bucket. This will reduce operating costs but this architecture has latency problems, which is why we made some changes to the MARLA Architecture to improve the system's performance. Fig 2 shows the MARLA architecture which consists of the three component the coordinator, mapper and reducer.

To improve the performance by keeping the architecture cost efficient we have modified the MARLA architecture with locus system, in which it combine the slow storage with cheap rate and fast storage with high rates and result in high performance, resource efficient and cost efficient.

4.1 Algorithm used:

In serverless platform, pricing is depend on the number of times does the function execute for required period of time. So, if the required period to execute the function is large then the pricing in the serverless platform is obviously high. In this scenario task scheduling plays vital role. We used genetic algorithms to plan the task and to find the best outcome. Since the execution time for the genetic algorithm is low, the time needed to perform the function is low and operation costs are low.

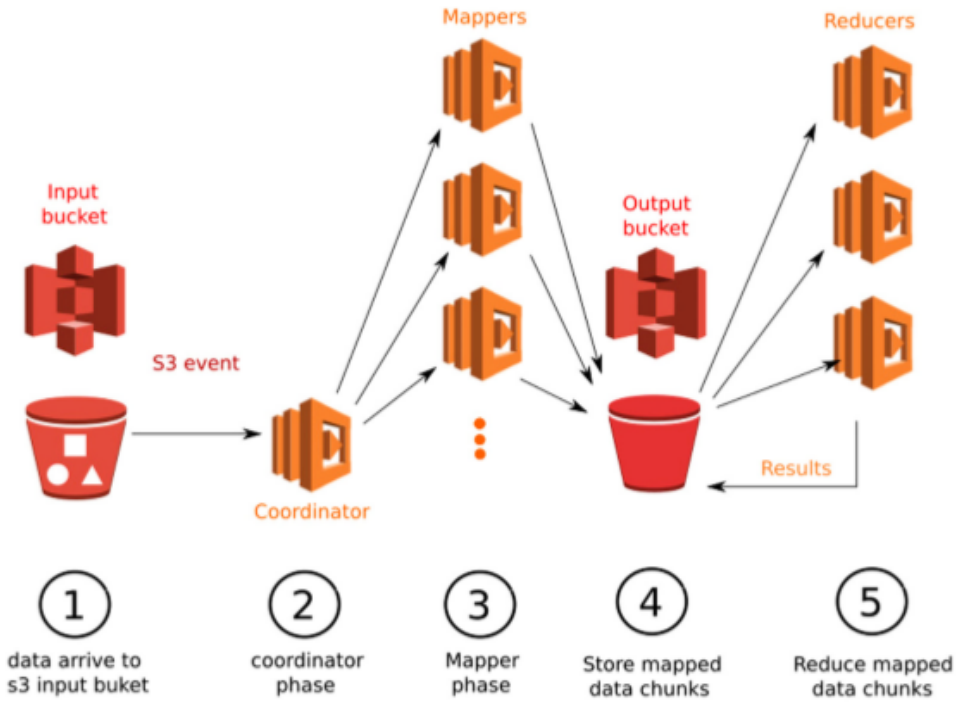

Figure 2: Architecture of Marla to support map reduce on aws lambda Giménez-Alventosa et al. (2019)

4.1.1 Genetic Algorithm:

The following process explains how a genetic algorithm works and the genetic algorithm pseudo-code shown in 1.

1. Genetic algorithm starts with the Initial population of related solutions to a problem.

2. Initial population is created with the help of chromosomes in randomly manner and chromosomes are in the form of string or character.

3. In the next step, fitness of individual is calculated withing the population. So, the result who satisfy the criteria is kept in population and the result who do not satisfying the criteria is Removed from population.

4. In a next step genetic operators are used. Initial one is selection operator, In this chromosomes who satisfies the fitness criteria are kept in a population and used for mating.

5. To generate the new off-springs crossover and mutation genetic operators are carried out on a selected parents.

6. In a crossover generator it swaps the chunks of data between two chromosomes and finds the population's diversity when small portion of population change is introduced.

7. The overall procedure is repeated until the function meets the criteria; when best solutions are found, the process ends and the best result comes back to the problem.

The below figure1 shows the pseudo of genetic algorithm.

Algorithm 1 Pseudo-code of Enhanced genetic algorithm

Input: mapreduce application
pop_size,
Mutation Probability,
request,
size of population,
file content;
Output: A task scheduling
1: call Initial Population;
2: repeat
3: call Selection operator;
4: call Crossover operator;
5: call Mutation operator;
6: call Inversion operator;
7: until size of population

4.2 Modified Architecture:

The Modified architecture consists of s3 bucket for input data, lambda function for mapper, reducer and coordinator, Redis storage for storing the intermediate data generated through mapper and another s3 bucket for storing the output generated through reducer. below figure 3 shows the modified architecture.

The operation starts when there is input data in s3, so it invokes the coordinator lambda function. The lambda coordinator function calculates data size by dividing total data size into a number of user chunks. this data is stored in a allocated RAM, but if the mapper don't have the enough RAM then coordinator again recalculates the data size for each mapper.

Initially it calculates the chunk size processed by mapper with the help of following formula,

total size of data = x
number of mappers = y

$$datachunksize = \frac{x}{y} \tag{1}$$

After this it checks the whether the chunk size satisfies the bellow conditions.

1. when the chunk size is lower than the minimum block size defined by the customer. So, the number of mapper size recalculated as,

$$y = int(\frac{x}{minblocksize}) + 1 \tag{2}$$

2. If the chunk size is larger than the safe memory size then chunk size is set to be as safe memory size and again the number of mappers are calculated as below,

$$y = int(\frac{x}{safememorysize}) + 1 \tag{3}$$

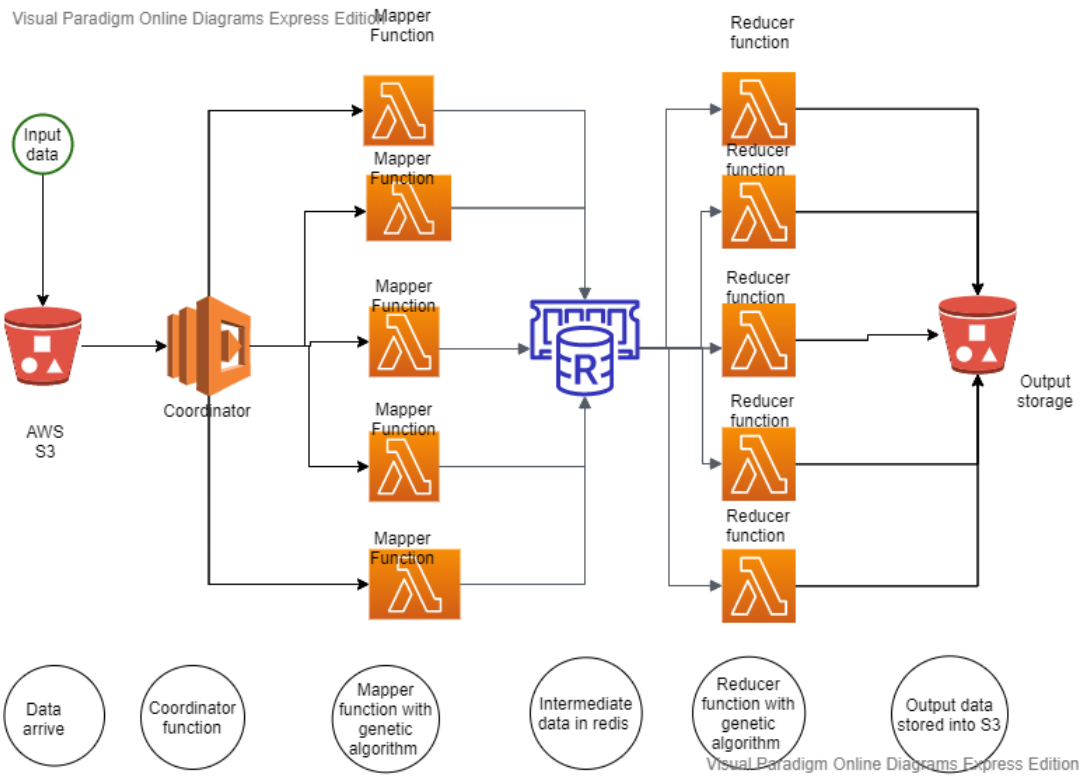

Figure 3: Modified architecture of map reduce on aws lambda

Safe memory size is nothing but the proportion of memory dedicated to mapper function.

3. If the chunk size larger than the max block size specified by the customer. So in this scenario chunk size is set to max-block size and the number of mappers are set to as,

$$y = int(\frac{x}{maxblocksize}) + 1 \tag{4}$$

In above all equation while calculating the mapper size we have added one extra mapper to process the residual data size: residualData = z

$$z = x - (y - 1).datachunksize \tag{5}$$

In the worst case scenario the size of the residualdata is number of mappers -1 so it avoids the mapper function to process chunk data plus the residual data. So, it invokes the initial mapper lambda function. In this it uses the logarithmic method to invoke the second mapper lambda function then the process continues. The data is distributed using the coordinator function and assigned data to mapper function. Then mapper function segregate the data using genetic algorithm and divide that data into a chunk of data and store that intermediate data into Redis storage. Lastly the intermediate data assigned to reducer and store the output data into s3 bucket.

5 Implementation

As we have seen AWS lambda provides the better scalability and has lower cold start delay than other cloud service provider so we have chosen the lambda service as computation engine for the project. To implement the genetic algorithm based MapReduce application on serverless platform following configuration we have created with using amazon services.

- To access multiple services such as s3, cloud watch etc, created a role whose name is lambdapermissions and assign a policies as mentioned in 4
- Created VPC (server_vpc) with 192.168.0.0/16 ipv4 CIDR block, public and private subnet with 192.168.10.0/24 192.168.20.0/24, public and private route table namely public_RT and private_RT. we did public subnet association to public route table and private subnet association to private route table. Created internet gateway (igw-08fede8e78a821f67) to access internet though public subnet, it allow the resources in your Virtual private network to access the internet and NAT gateway(nat-0d8b3828937c6afc5) enables private subnet resources to access the internet.
- Created the security group(sg-0660833480f433c5f - securityforvpc) which act as virtual firewall so it allows the incoming traffic and outgoing traffic. In this security group it allows the traffic from anywhere to Redis and other services with the help of HTTP, HTTPs, SSH, custom TCP, and All ICMP – Ipv4 protocol.
- Created s3 bucket(aws-trigger1) with all public access to store the incoming input data and VPC endpoint toMake private connections with VPC and s3 without internet access, by NAT, VPN or direct connection.
- Created the elastic cache for Redis with assigning the public, private subnet and security group to it to store the intermediate data to Redis.
- Created lambda function(Redis) by assigning IAM role, VPC along with security group also we have added s3 event trigger to lambda function so whenever the data comes into bucket it triggers the lambda function.

In this paper, we have used Amazon s3 for storing the input and output data, AWS Redis as middle storage and Lambda function as a compute engine. To run the map-reduce jobs on the serverless platform, we have used python 3.8 as a runtime programming language. In this, we have imported Boto3 to access S3 and Redis services, imported CSV file for reading CSV files, import json for converting data structures to json strings, import math function to use mathematical equation and Redis for cache memory functionality. In this, when the data comes into the s3 bucket, the s3 event triggers happen for lambda function execution. Initially, we have calculated the file size in Lambda and convert the file content into utf-8 format. After that, we need to calculate the chunk file size to assign the file size to mapper function equally, and it calculated by using,

$$chunksize = int\frac{filesize}{mappernumber} \tag{6}$$

But if the mapper has insufficient RAM, then the coordinator function recalculates the chunk size calculated by considering the two conditions mentioned in the methodology section. After this, assign the file size to a genetic algorithm as request along with mutating probability ranges between 0.2 to 0.5 and pop_size = 5. Then population

function performs along with these parameters; in this, it is started with the initial population generated randomly, performs the function evaluation, then selects the fittest individual as parents for next-generation and add some random non-fittest individual to avoid the local maxima using the cross over and mutation operator. Depending upon the pop_fitness value number of mapper decided. If the value of pop_fitness == 1, then one mapper function used, if it is 2, then two mapper functions used, and its result stored in Redis data storage. After that, one reducer function executed to sort out the data and the result stored in the s3 bucket.

So whenever the lambda function executed its output generated, and we can check it on cloud watch longs along with the cloud watch matrix. In a serverless platform, Price based on the number of services invoked at the function lambda. In this project, we have used s3 bucket, VPC, NAT gateway, Internet gateway, lambda, redis and cloud watch. So due to pricing constraints at education level, we have used one lambda function with 820MB memory, amazon s3 service, the elastic cache for Redis, but at the organization level, one can use separate lambda function for multiple mapper/reducer and coordinator.

6 Evaluation

The MapReduce application with a genetic algorithm on a serverless platform reduces the execution time than the MARLA system, and it is nearby of AWS EMR completion. The combination of fast and slow storage, along with a genetic algorithm, helps to minimize latency and improves efficiency and performance. The graph 4 shows the execution time to process the different file sizes.

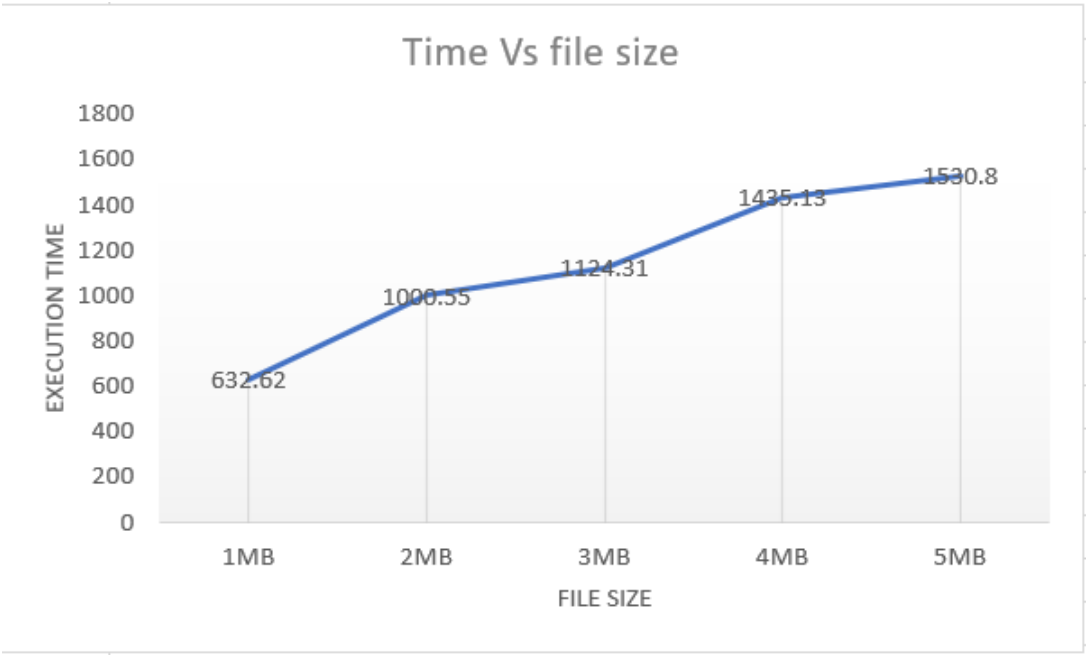

Figure 4: Time takes for file execution with genetic algorithm

6.1 Experiment 1: Map-reduce application without genetic algorithm on serverless platform.

In map-reduce application without genetic algorithm on serverless platform, we have uploaded a comma-separated values file with a 1MB file size. To execute this file size lambda function takes the 12484.60 ms. Figure 5 shows the output of map-reduce application without genetic algorithm on serverless platform.

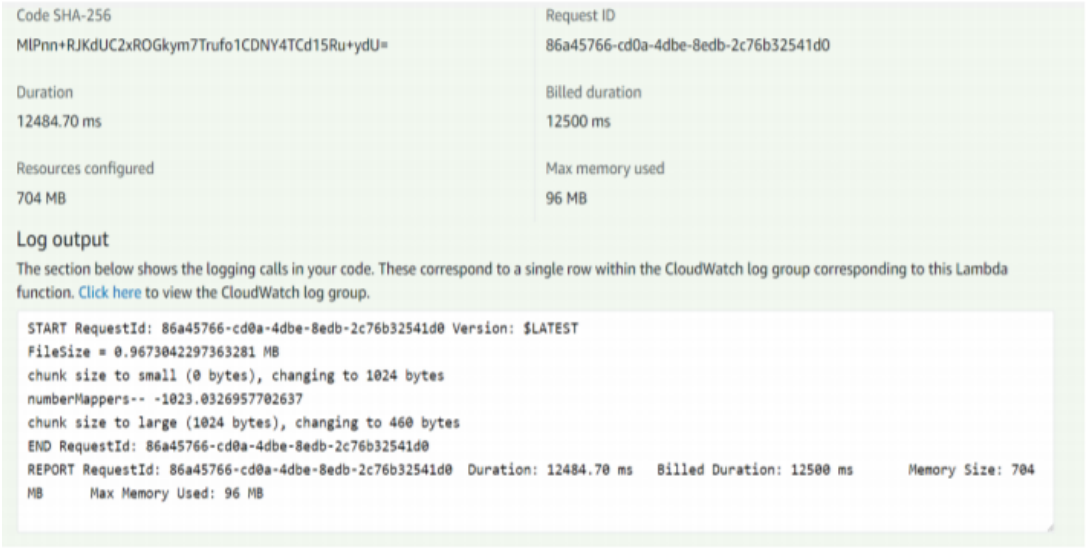

Figure 5: Executing 1 MB file size on serverless platform without genetic algorithm

6.2 Experiment 2: Map-reduce application with genetic algorithm on serverless platform

Initially, we have started by executing the file size with 1MB to check the amount of memory and time it takes to execute the file. We have uploaded a comma-separated file in the s3 bucket, so trigger event happens at lambda handler function, and the file executed. As a result, figure 6 shows 834.26ms time to execute the file, and the amount of memory also used genetic algorithm shows how many mappers we need to invoke to execute the file size. For 1MB file size, it invokes 1 mapper and takes the 3 generations to find out the fitness of the population. In output data, we have got the sorted data.

We have uploaded a file size of 2MB in a comma-separated values file, invokes 3 mapper function, and takes 5 generations to find out the best fit solution. Total time to execute this file size 4155 ms.

Now In this, we have uploaded the comma-separated values file size with 9 MB, and the lambda function has 6 MB maximum payload size. It gives the error to execute the file size. figure 7 shows error obtained while executing file greater than the maximum payload size.

In map-reduce application on the serverless platform with the genetic algorithm, we have executed a file size from KB to 5MB, and all file size data takes 3 seconds to perform the execution. Lambda uses VPC resources, so this creates an initial cold start with the elastic network interface (ENI). As in this project, we have implemented the coordinator,

Figure 6: Executing 1 MB file size on serverless platform with genetic algorithm

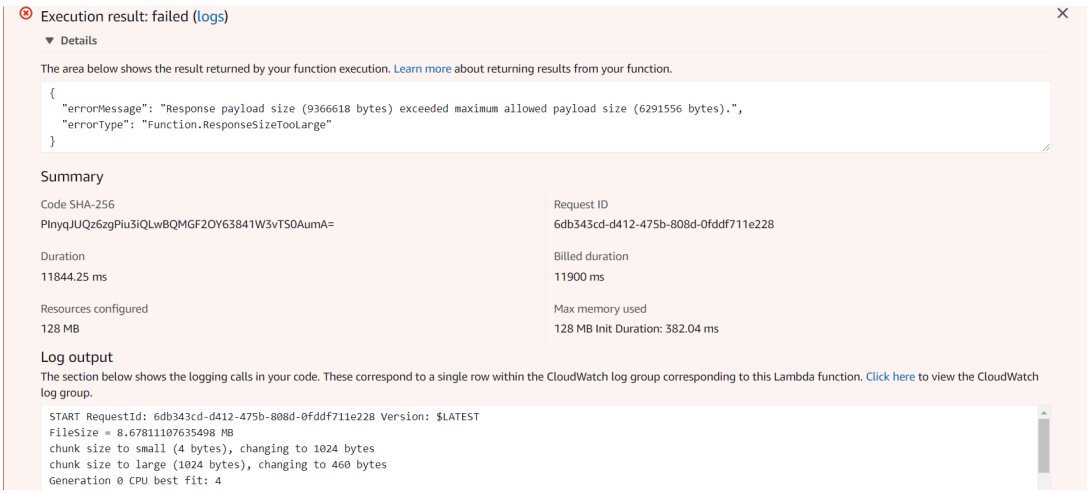

Figure 7: Executing 9 MB file size on serverless platform with genetic algorithm

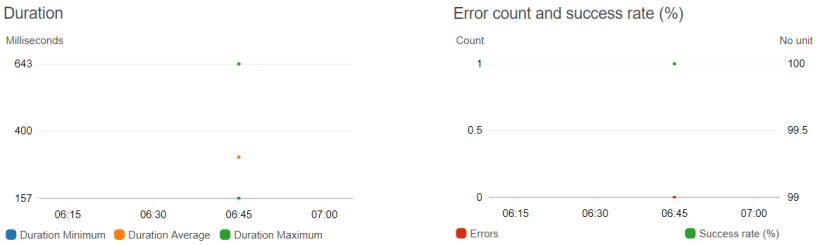

Figure 8: Duration, success and error rate

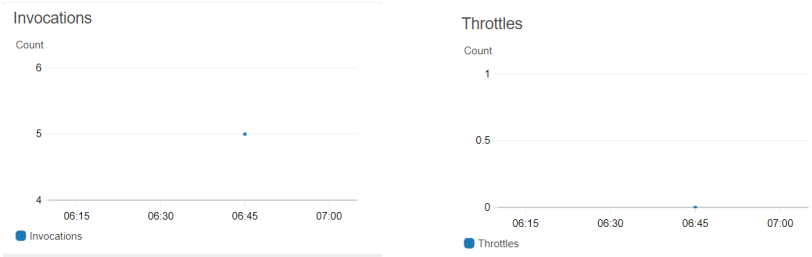

Figure 9: Invocation and Throttles graph

mapper, and reducer inside the single function, so there is a minimum cold start delay at the initial stage.

We have used cloud watch services to monitor duration, error count, throttle time, and invocation time graph. The duration graph 8 describes at what time the coordinator function starts to execute the file and total time take to complete the execution. The green dot used to describe the maximum time it takes to execute the file. The error count and success rate graph 8 describes what the success rate is; in this green dot represent the 100% data executed without any data.

The invocation graph 9 describes the number of times the function invoked. In this project s3 event is used to invoke lambda function, and according to file size, it invokes the mapper function. In this case, it is 5. In AWS Lambda function has 1000 limits for maximum concurrent operations, and we are not using any concurrent function, so it is giving a zero throttle rate. figure 9 show the throttle graph.

Now working with Redis, In a project, we have used the cache.t2.micro node, which has 1 vcpu, so the threshold value is 90%, and it is under the threshold value.

The network bytes in and out shows that the data are entered and released by Redis. Also, it shows that data goes in and out equally with 100% data utilization.

6.3 Discussion

We have executed file size from 1MB to 5MB on map-reduce application without a genetic algorithm on the serverless platform and with a genetic algorithm. The comparison based on execution time in both cases vs. file size. As shown in graph 10 map-reduce application with the genetic algorithm on a serverless platform requires less execution time as compared to the map-reduce application without a genetic algorithm.

1MB file size requires 703.13 ms execution time in MapReduce application with a genetic algorithm and 1271.01ms execution time in MapReduce application without the genetic algorithm. In AWS, Nat gateway requires $0.045 per hour, s3 storage requires $0.023 per GB, and Redis(cache.t2.micro) requires $0.017 per hour pricing. For 703.13

ms execution time, the total cost is $0.000038, and for 1217.01 ms, it's cost is $0.000047. So, in this application with a genetic algorithm reduces the cost by 11%. The MapReduce application with a genetic algorithm reduces the execution time by 27% than the MapReduce application without the genetic algorithm. Also, we have used a combination of fast and slow storage in map-reduce application with a genetic algorithm, so it gives higher performance with a reduction in cost.

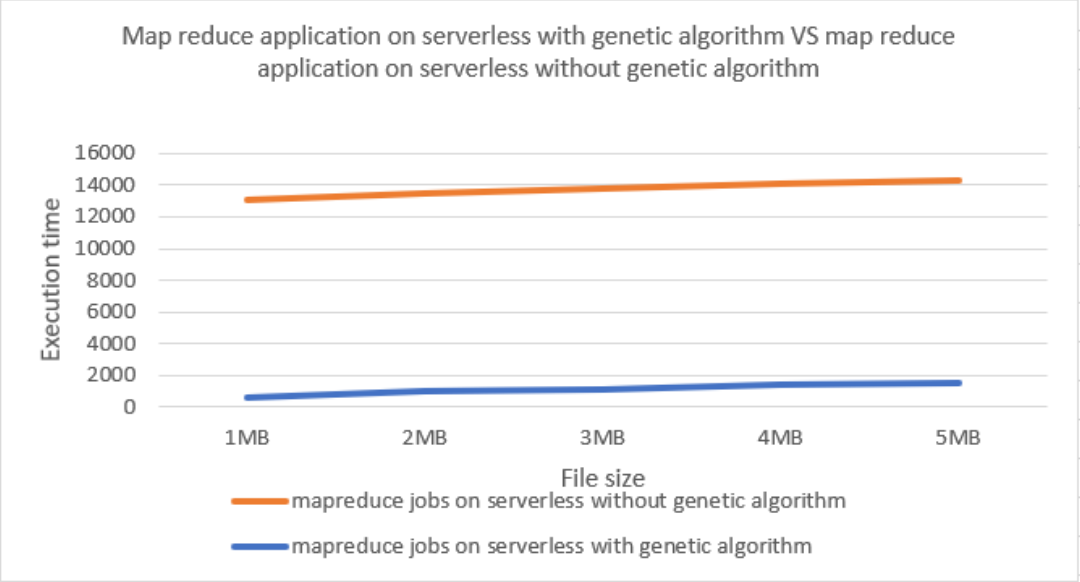

Figure 10: Comparison between map-reduce application with genetic algorithm vs. map-reduce application without genetic algorithm on serverless platform

7 Conclusion and Future Work

In this paper, we have proposed the Map-reduce application with the genetic algorithm on a serverless platform. This paper states that the makespan of the MapReduce application reduced by using the enhanced genetic algorithm. The combination of fast storage(Elastic cache for Redis) and slow storage(S3 standard storage) reduces the latency and improves the performance of the operation. The MapReduce application with a genetic algorithm reduces the execution time by 27% than the MapReduce application without a genetic algorithm as it invokes the number of mappers according to the fitness of the population; ultimately, it reduces the cost of the operation. AWS cache for Redis used for intermediate storage provides a reduction in latency but increases the cost. Due to the limitation of resources, this model presented for a small platform. Future work consists of enlarging the MapReduce application with a genetic algorithm model to a big platform that consists of creating clusters of lambda function and elastic cache in amazon elastic cloud compute service and implementing it in another platform as well.

References

Akbari, M., Rashidi, H. and Alizadeh, S. H. (2017). An enhanced genetic algorithm with new operators for task scheduling in heterogeneous computing systems, **61**: 35–46.

journal impact factor : 2.819.
URL: *http://www.sciencedirect.com/science/article/pii/S0952197617300441*

Arunarani, A., Manjula, D. and Sugumaran, V. (2019). Task scheduling techniques in cloud computing: A literature survey, **91**: 407–415. journal impact factor:4.639.
URL: *http://www.sciencedirect.com/science/article/pii/S0167739X17321519*

Asghari, S. and Navimipour, N. J. (2018). Nature inspired meta-heuristic algorithms for solving the service composition problem in the cloud environments, **31**(12): 1–1. Publisher: John Wiley & Sons, Inc., journal impact factor : 1.717.
URL: *http://search.ebscohost.com/login.aspx?direct=trueAuthType=ip,cookie,shibdb=a9hAN=13050*
livescope=sitecustid=ncirlib

Buyya, R., Srirama, S. N., Casale, G., Calheiros, R., Simmhan, Y., Varghese, B., Gelenbe, E., Javadi, B., Vaquero, L. M., Netto, M. A. S. and et al. (2018). A manifesto for future generation cloud computing: Research directions for the next decade, *ACM Comput. Surv.* **51**(5). journal Impact Factor : 5.550.
URL: *https://doi.org/10.1145/3241737*

Costa, A., Cappadonna, F. V. and Fichera, S. (2020). Minimizing makespan in a flow shop sequence dependent group scheduling problem with blocking constraint, **89**: 103413.
URL: *http://www.sciencedirect.com/science/article/pii/S0952197619303227*

Enes, J., Expósito, R. R. and Touriño, J. (2020). Real-time resource scaling platform for big data workloads on serverless environments, **105**: 361–379. journal impact factor:4.639.
URL: *http://www.sciencedirect.com/science/article/pii/S0167739X19310015*

Gawanmeh, A., Parvin, S. and Alwadi, A. (2018). A genetic algorithmic method for scheduling optimization in cloud computing services, **43**(12): 6709–6718. Publisher: Springer Nature, core ranking : C.
URL: *http://search.ebscohost.com/login.aspx?direct=trueAuthType=ip,cookie,shibdb=a9hAN=13290*
livescope=sitecustid=ncirlib

Giménez-Alventosa, V., Moltó, G. and Caballer, M. (2019). A framework and a performance assessment for serverless MapReduce on AWS lambda, **97**: 259–274. journal Impact Factor: 4.639.
URL: *http://www.sciencedirect.com/science/article/pii/S0167739X18325172*

Jena, T. and Mohanty, J. R. (2018). GA-based customer-conscious resource allocation and task scheduling in multi-cloud computing, **43**(8): 4115–4130. Publisher: Springer Nature, core ranking : C.
URL: *http://search.ebscohost.com/login.aspx?direct=trueAuthType=ip,cookie,shibdb=a9hAN=13055*
livescope=sitecustid=ncirlib

Liroz-Gistau, M., Akbarinia, R., Agrawal, D. and Valduriez, P. (2016). FP-hadoop: Efficient processing of skewed MapReduce jobs, **60**: 69–84. journal Impact factor = 4.313.
URL: *http://www.sciencedirect.com/science/article/pii/S0306437916300497*

Luo, J., El Baz, D., Xue, R. and Hu, J. (2020). Solving the dynamic energy aware job shop scheduling problem with the heterogeneous parallel genetic algorithm, **108**: 119–134.
URL: *http://www.sciencedirect.com/science/article/pii/S0167739X19314189*

Malawski, M., Gajek, A., Zima, A., Balis, B. and Figiela, K. (2020). Serverless execution of scientific workflows: Experiments with HyperFlow, AWS lambda and google cloud functions, **110**: 502–514. journal impact factor= 4.639.
URL: *http://www.sciencedirect.com/science/article/pii/S0167739X1730047X*

Marra, M., Polito, G. and Gonzalez Boix, E. (2020). A debugging approach for live big data applications, **194**: 102460.
URL: *http://www.sciencedirect.com/science/article/pii/S0167642320300708*

Pu, Q. (Feb 2019). Shuffling, fast and slow: Scalable analytics on serverless infrastructure, p. 15.
URL: *https://www.usenix.org/system/files/nsdi19-pu.pdf*

Rajeswari, D., Prakash, M. and Suresh, J. (2019). Computational grid scheduling architecture using MapReduce model-based non-dominated sorting genetic algorithm, **23**(18): 8335–8347. Journal Impact Factor : 2.367.
URL: *https://doi.org/10.1007/s00500-019-03946-z*

Ruiz, M. C., Cazorla, D., Pérez, D. and Conejero, J. (2016). Formal performance evaluation of the map/reduce framework within cloud computing, **72**(8): 3136–3155.
URL: *http://link.springer.com/10.1007/s11227-015-1553-2*

Sathya Sofia, A., Emmanuel Nicholas, P. and Ganeshkumar, P. (2019). MCAMC: minimizing the cost and makespan of cloud service using non-dominated sorting genetic algorithm-II, (10): 1. Publisher: Springer, journal impact factor : 0.295.
URL: *http://search.ebscohost.com/login.aspx?direct=trueAuthType=ip,cookie,shibdb=edsgaoAN=eds*
livescope=sitecustid=ncirlib

Sun, M., Zhuang, H., Li, C., Lu, K. and Zhou, X. (2016). Scheduling algorithm based on prefetching in MapReduce clusters, **38**: 1109–1118. Journal Impact Factor: 3.907.
URL: *http://www.sciencedirect.com/science/article/pii/S1568494615002665*

Wang, L., Li, M., Zhang, Y., Ristenpart, T. and Swift, M. (2018). Peeking behind the curtains of serverless platforms, p. 13.
URL: *https://www.usenix.org/system/files/conference/atc18/atc18-wang-liang.pdf*

Zhang, X., Liu, X., Li, W. and Zhang, X. (2019). Trade-off between energy consumption and makespan in the mapreduce resource allocation problem, *in* X. Sun, Z. Pan and E. Bertino (eds), *Artificial Intelligence and Security*, Lecture Notes in Computer Science, Springer International Publishing, pp. 239–250.
URL: *https://sci-hub.tw/10.1007/978-3-030-24265-7*$_2$1