National College of Ireland

MARLA Architecture On Azure Services

MSc Research Project
Cloud Computing

Janit Pathak
Student ID: 20186169

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:	Janit Pathak
Student ID:	20186169
Programme:	M.Sc. Cloud Computing
Year:	2022
Module:	MSc Research Project
Supervisor:	Vikas Sahni
Submission Due Date:	14/08/2022
Project Title:	MARLA Architecture On Azure Services
Word Count:	4539
Page Count:	19

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.

Signature:	Janit Pathak
Date:	15th September 2022

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):	

MARLA Architecture On Azure Services

Janit Pathak
20186169

Abstract

In cloud computing, the Function as a Service (FaaS) is significantly changing the design and architecture of the software services. The serverless computing provides freedom to user from managing hardware infrastructures which is required to host the software applications. Along with that, it also provides higher elasticity and scalablity to the application to provide fine-grained cost model. The Microsoft Azure is the one of the fastest growing cloud service provider but there is limited research available on MapReduce application fesiablity on Azure serverless infrastructure. The MapReduce application is a decentralized big data processing framework and it has capability of processing huge amount of data parallely. This paper implements MapReduce application on Microsoft Azure cloud computing services. It implemented MapReduce application by using Azure serverless computing and blob storage and it also provides evaluation of performance of the application by considering various parameters such has makespan, CPU utilization etc. Finally it provides the comparative analysis of implemented application with the other available MapReduce solutions. The research found that MARLA architecture on AWS has shorter makespan than Azure.

1 Introduction

With the changing time the worth of different commodity keeps getting changed. In the current century the data is classified as the most precious commodity. In last 2 decades it is observed a great change led by digital revolution. The use of IT has been increased in every sector of industry. The increase of digitization results in the increase of the generation of data. This comes with the challenge of managing huge amount of data. The MapReduce application is one of the most prominent application for processing big data. Cloud has made the implementation of different application independent of their hardware considerations. The services offered by cloud provider such as Platform as a Service(PaaS), Software as a Service(SaaS) etc. has made it easy to implement solutions which are scalable and highly available. But it is extremely important to optimize resource utilization on cloud because it has pay-as-per-usage model.

This project consists of two stages: mapping tasks and reducing tasks. Enormous data is divided into smaller chunks and processed in parallel with the aid of map and reduce workers. One of the biggest drawbacks of MapReduce systems is their poor task scheduling. Hadoop 2.0 enhances resource management for MapReduce tasks by introducing the cluster resource manager. Although the present Hadoop job scheduler still needs improvement.

The rate of data production per day has surpassed petabytes in the realm of new internet technologies. Data management, data analytics and data processing have become key

difficulties and the requisite infrastructure, operating costs, storage and security are the primary bottlenecks in addressing these issues. Serverless cloud computing may be a viable option for addressing current bottlenecks and providing a cost-effective solution. The serverless platform uses a pay-per-use pricing model because of which a non-optimized MapReduce application with poor job scheduling would not only be inefficient, but also expensive.

In this research paper, the existing MARLA architecture is implemented by using Microsoft Azure services which is made to increase data locality and deal with latency issue in the existing MapReduce architecture implementation. Serverless computing services in azure is being used for implementation and resultant data is stored on Azure blob storage. Later, an extensive discussion is made for detailed performance analysis of implemented architecture and finally a descriptive comparison of MARLA(Mapreduce on AWS Lambda) architecture on AWS and Azure is carried out.

1.1 Research Question

Is the makespan of MARLA architecture is better in AWS or in Azure?

1.2 Research Objectives and Contributions

- Implementing MARLA architecture on Microsoft Azure Cloud.
- Evaluating its performance matrix
- Comparing its makespan with AWS MARLA architecture.

The contribution consists of:

- Critically analyzing and reviewing the existing research work performed in the identical area.
- Designing and implementing MARLA architecture on Azure.
- Designing the configuration manual to reproduce the work for public use.

2 Related Work

In this section the research work carried out in the area of optimizing MapReduce application is thoroughly examined. Additionally, multiple different architecture and implementation of MapReduce application on serverless platform is brought into consideration.

The below content is divided into following sections, in section 2.1 multiple works on improving MapReduce application are discussed. The Section 2.2 discusses an account of serverless architecture and the matrix of the analysis of available literature work and in Section 2.3 a discussion on implementation of MapReduce application on serverless platform is done.

2.1 Review on improved MapReduce Application

Hadoop MapReduce framework processes big data with high throughput but it has latency issue for processing short jobs. In the paper Gu et al.Gu et al. (2014) a discussion is made to reduce turn-around time by implementing critical event messages and reducing delay between submitting job and scheduling its task. Cleanup task has been optimized by saving 4 heartbeat intervals required in standard flow. An event notification mechanism is implemented which separates critical messages from normal heartbeat messages. This paper adopted the traditional heartbeat communication mechanism for events belonging to cluster management that are not performance-sensitive. It also used an instant messaging communication mechanism for messages belonging to critical events that are sent to JobTracker immediately so that message synchronization between JobTracker and TaskTracker occurs with less latency. The primary drawback of this approach is that, it overwhelms JobTracker and if the jobs are lengthy, the optimization doesn't yield much of an advantage.

Another bottleneck of the efficient MapReduce application is shuffling phase of the application. In the paper by Nikitas et al.Nikitas et al. (2021) a distributed task-aware shuffle-service for serverless analytics is implemented. It includes task-aware, look-ahead caching on remote storage to enhance I/O operations, stateless workload execution that provides fault tolerance for data loss or worker node failure and a modular architecture that makes system integration simple. A realistic and synthetic workload is used to evaluate the systems and parameters including completion time, fault tolerance, scalability, resource efficiency and data skews are included in the evaluation matrix.

2.2 Review on Serverless Architecture

Lee et al.Lee et al. (2018) have examined the throughput and performance of several cloud service providers while implementing serverless computing offerings for distributed data processing in parallel. In this paper, services are compared using ten matrices including CPU performance, performance after memory and disk-intensive functions, performance after concurrent and sequential invocations, etc. A constant increase in throughput is exhibited by throughput performance of Google Functions whereas IBM OpenWhisk [1] and Microsoft Azure Functions showed their best behavior at 2000 invocations and 2000 invocations respectively. The paper compared triggers, HTTP, databases, object storage and concluded that trigger performance is unaffected by trigger invocation.

Saha et al.Saha and Jindal (2018) have focused on improving resource management in their paper by introducing EMARS (Efficient Management and Allocation of Resources in Serverless) framework. It has used a workload and memory based predictive model to improve performance. The major finding of the paper is the improvement of memory. It is determined that memory allocated to the functions must be handled well in order to increase performance because the latency decreases and becomes saturated after a certain quantity of memory. In EMARS, a memory is allotted based on predictive models, logs are based on various parameters and they are recorded to a configuration file after a predetermined amount of time.

The main issue with serverless architectural performance is cold start of serverless functions. To address this issue, public cloud providers like AWS attempt to begin each function in a warm state. If all containers are already full, a new container is generated.

[1] url:https://openwhisk.apache.org/

In the paper by Gunasekeran et al.Gunasekaran et al. (2020) a presentation of Fifer is made which reduces container spawning through request batching and queuing and also reduces service level goal violation leading to more accurate load forecasting by making use of this model. Estimated execution time calculation and slack estimation for load balancing by dynamic reactive scaling policy, function scheduling and bin-packing to maximize resource use are the primary elements of the suggested design.

In the paper by Witte et al.Witte et al. (2020) an architecture which is capable of processing high computation intrinsic workloads on serverless platform is implemented. Usage of MPI to process the code in parallel loops is done. The AWS step function is used to deploy each software component individually using a generic optimization method. It creates multiple gradient of specific batch size to perform the computation. The batch queue receives parallel workload additions which AWS batch processes by launching an EC2 instance as needed. Each gradient's output is stored in an S3 bucket and AWS Lambda function is used to combine the results into a single gradient. The S3 bucket then sends an object to a SQS queue which then initiates the reduction stage. The fundamental burden of serverless and event-driven compute workflow is demonstrated in this paper. The biggest difficulty with this strategy is creating code that can be broken down into a number of distinct modules.

AWS, Azure, and other commercial solutions are widely used but free source alternatives can potentially deliver the desired effects. In the paper Djemame et al.Djemame et al. (2020) have offered information on the viability of adopting Apache OpenWisk as an open-source option for serverless architecture. The bench-marking and performance analysis of the Apache OpenWhisk platform is made to utilize a set of test functions and performance analysis is carried out taking into account various criteria such as efficiency and effectiveness as cloud-based solutions. The study demonstrates that OpenWhisk has the ability to outperform a system that uses container-based virtualization and has equivalent functionality, as well as it can deliver a superior performing solution without virtualization overheads. The fundamental flaw in this study is that parallel computing was not taken into account when creating its benchmarks and quality of service and cost comparison should also have been included to give a more accurate picture of the viability of the research work.

2.3 Review on Map-reduce application on serverless platform

The Giménez-Alventosa et al. (2019) et al. have provided framework to implement MARLA (MapReduce on AWS Lambda). The framework is created by using AWS lambda and AWS S3 bucket. The AWS lambda function performs mapping and reducing tasks and S3 bucket stores the data and triggers event to start the process. The MARLA framework has low latency during uploading of larger files. This paper has used other AWS services such as AWS Elastic Cache to reduce the latency and Approx algorithm is used to improve task scheduling which reduces the latency of the framework.

The MapReduce applications are scalable, fault-tolerant and capable of processing large amount of data but the longer execution time required for processing the large amount of data is a major challenge. Kalavri et al.Kalavri and Vlassov (2013) in their paper discussed different optimization approaches in MapReduce framework by giving different optimization technique such as operator pipelining and operator aggregation, approximated results, indexing and sorting, work sharing, data reuse, skew mitigation and data co-location. The batch nature of the incoming data is one of the major prob-

lems with the MapReduce programming model and the study conducted in this paper provided evaluation of a solution offered by Hive and Pig Latin to get around this restriction by enabling users to create SQL-like scripts. Brief accounts of studies relating to the MapReduce system are discussed and it is concluded that, in contrast to standard MapReduce programs, these systems are computationally intensive rather than data-dense and they have in-memory processing to minimize I/O operations.

Sampe et al.Sampé et al. (2018) have suggested new architecture to run MapReduce jobs on IBM Cloud Functions and PyWren. IBM Cloud Object Storage (IBM COS) has been employed for data storage and IBM Cloud Functions is used for MapReduce workloads and function compositions. When an event occurs, a certain function is called by an IBM function and the function code is fetched from ICOS. This is an event-driven technique. The system's API offers a variety of parallel processing techniques including call async(), map(), map reduce(), wait() and get result(). By serializing the function code and input data, parallel code is executed. To obtain the necessary data for data execution, a HEAD request is made via the ICOS bucket. Partitioning is then carried out in accordance with the configuration file. The partitioned data is then sent to the executor function which runs the map function and stores the results in ICOS. The research paper has expanded support for MapReduce jobs and other distributed computing processes and it is capable of running extremely concurrent operations in the IBM Cloud. With varying chunk sizes, the speedup for MapReduce execution can reach 135.79x.

2.4 Research Niche

The below table provides summary of the previous studies, it gives an account of methodology, advantages and gap analysis with respect to this research paper.

Table 1: Summary of Literature Review and Research Niche

Paper Title	Methodology	Advantages	Gap Analysis
Lee et al. (2018)	Comparison matrices to assess the effectiveness of serverless computing provided by various cloud service providers	compared many aspects and the ones that have no impact on performance	Processing Big-Data is not considered
Saha and Jindal (2018)	workload-based and memory-based predictive models for Efficient Management and Allocation of Resources in Serverless	limit high memory utilization and predictive model	Container based approach

Paper Title	Methodology	Advantages	Gap Analysis
Gunasekaran et al. (2020)	Adaptive resource management framework to efficiently manage function-chains on serverless platforms	Cold start problem	Limited scalability because of centralized database
Witte et al. (2020)	Serverless and event-driven approach for data intrinsic computation	Nested level of task paralleling	Cost of operation is high
Djemame et al. (2020)	Serverless architecture using Apache OpenWisk as an open-source solution	A better performance solution sans virtualization overheads uses container-based virtualization.	Parallel computing is not included
Kalavri and Vlassov (2013)	Mapreduce: Limitations, optimizations and open issue	Account of various methods for optimizing the MapReduce framework	Just concentrated on decreasing I/O operations and computation-intensive tasks.
Giménez-Alventosa et al. (2019)	MARLA(MapReduce on AWS Lambda) architecture	Application MapReduce was implemented using AWS Lambda	Inefficient task Scheduling
Gu et al. (2014)	Hadoop framework improvisation through task and job execution optimization	support job scheduling improvements to enhance job performance	Long-term projects render a framework ineffective.
Nikitas et al. (2021)	Serverless Analytics distributed task-aware caching shuffle service	Data loss or worker node failure fault tolerance	Only the Reduce stage should be optimized, as the Map stage requires the greatest processing.

Sampé et al. (2018)	MapReduce Jobs on PyWren and IBM Cloud Function	Framework execute massive parallel tasks and distributed computing processes	Makespan is high as compared to other framework with AWS services
THIS ONE	Implementation of MARLA architecture on Azure	Reduced makespan	Azure Cloud service dependent for improved performance

3 Methodology

MapReduce application by using Microsoft Azure Services has been implemented in this paper. Serverless computing services of Azure and Azure Blob storage has been used for implementing the Map Reduce application. Serverless platform Azure monitor has been used for monitoring the details related to execution of the application.

3.1 Steps:

There are three key stages in the application

1. Mapper
2. Coordinator
3. Reducer

Azure function and Azure blob storage has been majorly used in the project to implement a complete serverless MapReduce application.

1. The Coordinator function is executed on Azure Function and the data is stored in Azure Blob storage. The process is event-driven; when new data is added to blob storage, an event is generated and the coordinator function is used to retrieve the dataset and divides it into segments based on the amount of new data. The serial number assigned to the data chunk serves as the key and the data content serves as the value in a series of key-value pairs that are distributed among mapper functions.
2. The coordinator function is used to invoke mapper function which computes the desired operation on the data chucks. In this case, the data is arranged in alphabetical order. This process gets repeated until all the data gets processed. Reducer bucket is used by all the mappers to store data and use it intermediately. The intermediate data is sorted and these data chunks are tagged with the partition key which is later used by reducer to consolidate the chunks.
3. Following the mapping of all the data chunks, the list of intermediate key-value pairs is provided to the reducer based on the key of each pair. Each reducer is used to add up all the samples and provide the dataset with same key number. By computing the mean value of the total, a new cluster center is created and the final result is saved in reducer blob storage.

The Figure 1 shows the execution workflow of the mapreduce job.

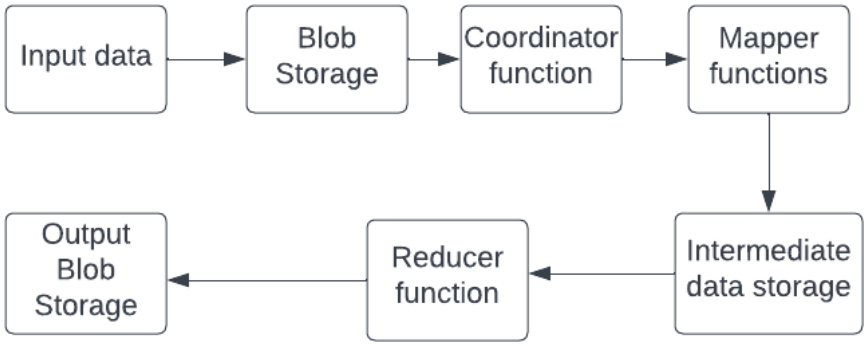

Figure 1: Application workflow

3.2 Material and equipment

3.2.1 Azure Services

Azure Active Directory Various Azure services access is required to perform desired function in this paper. In order to work with the desired Azure services it is needed to have an user with full access to Azure Functions and Azure Blob Storage.

Azure Blob Storage It can store static and dynamic data with the capability of scaling according to the requirements[2]. This paper uses two blob storages in which one storage is input blob where the dataset gets uploaded and it also triggers the process by invoking coordinator. The another blob storage is used to store the intermediate and output from reducer.

Azure Functions It enables users to run code without setting up and maintaining infrastructure. It offers a pay-as-you-go pricing structure and can scale automatically based on demand[3]. For the coordinator, mapper and reducer functions in this paper, Azure function services are used. The mapper function processes the data and distributes the data to the reducer function which aggregates the output produced by the mapper function.

3.3 Programming language

The programming language used in this paper is Python 3.8 to create functions and use Shell script for creating infrastructure.

3.4 Sample Data

The data used in the MapReduce function is based in the United States (public dataset). It consists of proper count of the number of people in the comma separated file without an email ID (CSV). The map function processes the input data and uses a key-value pair as a reducer to classify the pertinent data from the given data collection. The reducer

[2]`https://azure.microsoft.com/en-us/services/storage/blobs/`

[3]`https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview`

function uses key-value pairs to process the data and stores the results in an S3 bucket. Sample Data: `https://briandunning.com/sample-data/`

4 Design Specification

The presented paper acts as an extension of the research paper presented by Giménez-AlventosaGiménez-Alventosa et al. (2019)et al. which proposes MARLA architecture Figure:2 to execute python-based MapReduce jobs on AWS Lambda. The architecture is created entirely within the AWS cloud structure without the need of any outside components. The coordinator, reducers and mappers are the three serverless computing groups that have employed AWS Lambda functions. Additionally, it also used two AWS S3 buckets to store processed and intermediate data. Even though the MARLA architecture makes use of AWS cloud serverless services but it is limited to AWS Cloud services, the performance and feasibility of MARLA architecture on other cloud platforms needs to be explored.

The prospect of implementing MARLA architecture on other cloud platform will provide better understanding of its behaviour on other cloud platforms and its cost implications. The presented paper implements the existing MARLA architecture on Azure cloud platform.

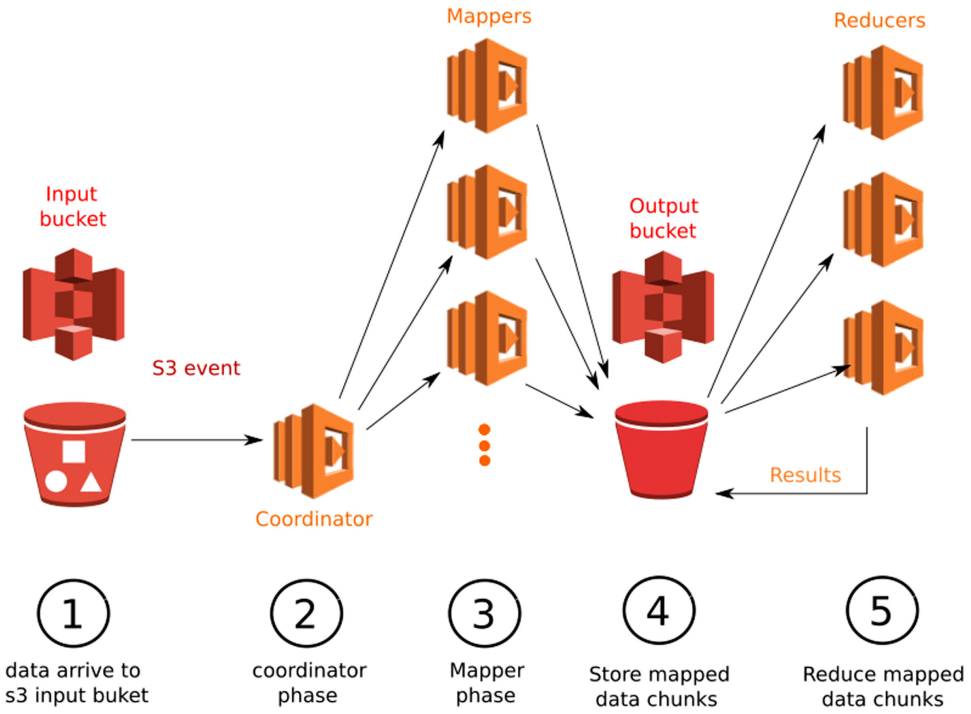

Figure 2: MARLA architecture presented by Giménez-AlventosaGiménez-Alventosa et al. (2019) et al.to support MapReduce on AWS Lambda

4.1 Modified Architecture:

The presented paper has implemented MARLA architecture on Azure Services to observe its behaviour. Changes in MARLA architecture has been made to reduce data latency

issue due to slow storage. The modified architecture has Azure blob storage for input and output data, azure function for mapper, reducer and coordinator. The figure 3 shows the modified architecture.

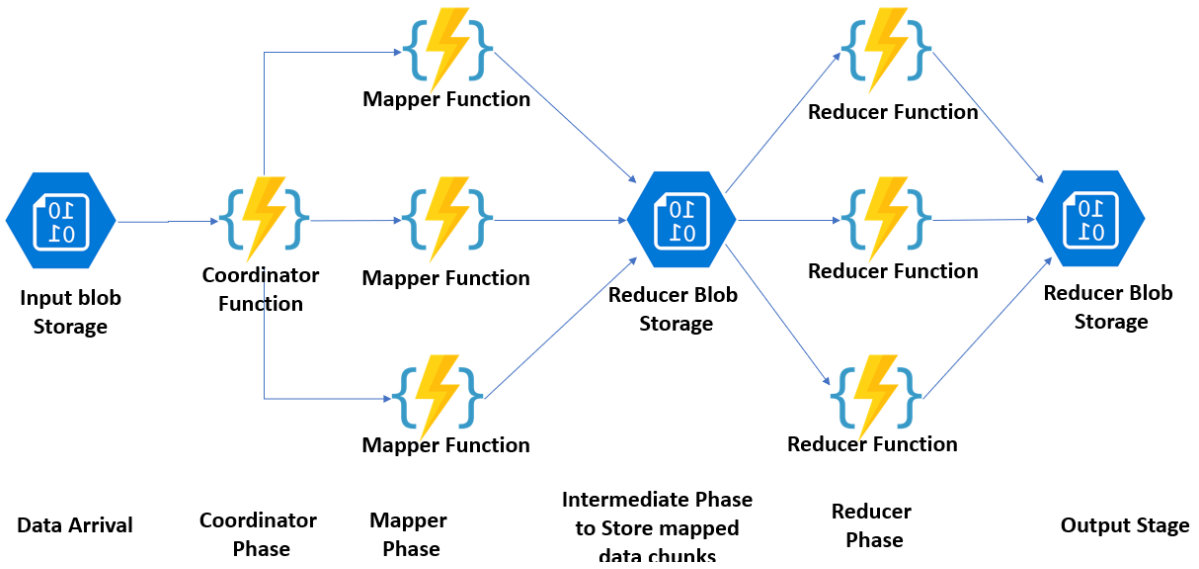

Figure 3: Modified architecture to support MapReduce on Azure

When data is placed in input blob storage, the process is triggered and coordinator azure functions are called. Depending on the size of the data, the coordinator functions break it into smaller pieces. The amount of accessible mappers affects the quantity of data chunks. The coordinator Lambda function re-evaluates the number of chunks so that it can fit into each mapper in case the RAM at the mappers is insufficient.

To calculate the chunk size, the coordinator function divide total size of data by the number of mappers.

Total Data Size = D_t

Number of mappers = N_{mapper}

$$ChunkSize = D_t / N_{mapper} \tag{1}$$

Then the coordinator function evaluates the feasibility of the chunk size by the following conditions:

1. If the *chunkSize* is smaller than the minimum block size defined by user(MINSIZEOFBLOCK).

$$N_{mapper} = int\left(\frac{D_t}{MINSIZEOFBLOCK}\right) + 1 \tag{2}$$

2. If the *chunkSize* is bigger than safe memory size(SafeSIZEofMemory).

$$N_{mapper} = int\left(\frac{D_t}{SafeSIZEofMemory}\right) + 1 \tag{3}$$

3. If the chunk size is bigger than the maximum block size then,

$$N_{mapper} = int\left(\frac{D_t}{MAXSIZEOFBLOCK}\right) + 1 \tag{4}$$

In above recalculations of N_{mapper}, coordinator function has added one extra mapper to process the residual data chunk with size as below. This prevents overloading in the mapper by processing residual data.

$$residualData = D_t - (N_{mapper} - 1).chunkSize \tag{5}$$

Then the coordinator invokes the first mapper with variable such as data chunk size. The mapper function invoke other mapper function in logarithmic reduction($log_2(N_m)$) manner until all mappers get invoked. Each mapper then processes the given chunk; this is the mapping phase. A list of reduced key-value pairs where reduction is carried out in the following step is the output of the map operation. In-memory storage is used to keep the intermediary data. Then, separate reduce functions process the mapper-processed chunks, processing all of the data using the same key in each reduce function. To accomplish this, reducers download as many mapped chunks as are compatible with the allocated amount of RAM. Once all of the data chunks have been handled by reducer, this procedure is repeated using the previously processed chunks and the newly processed chunks. The output is finally saved in blob storage.

5 Implementation

The presented paper acts as an extension of the MARLA architectureGiménez-Alventosa et al. (2019), it is a python based MapReduce framework for serverless cloud computing framework. To implement the MARLA framework of Azure, the following configuration is performed.

- Creation of an Active Directory user by assigning access to multiple services such as Azure function and Azure Blob Storage.
- In VPC, creation of CIDR block of 192.168.0.0/16, 192.168.10.0/24 and 192.168.20.0/24 to enable access to the internet using public subnet.
- Creation of the Network Security group which allow incoming and outgoing traffic.
- Creation of an Azure blob storage which is privately connected to VPC.
- The configuration setup of the file according to the execution requirement. Following are the parameters which are specified necessarily.
  - ClusterName: Name of the cluster
  - FunctionsDir: The directory containing the file that defines the Mapper and Reduce functions.
  - FunctionsFile: The name of the file with the Mapper and Reduce functions.
  - Region: The Azure region where the Azure functions is be created.
  - BucketIn: The bucket for input files. It must exist.

- BucketOut: The bucket for output files. There is need for different buckets for input and output to avoid unwanted recursions.
- RoleARN: The ARN of the role under which the Lambda functions is be executed.
- MapperNodes: The desired number of concurrent mapper functions.
- MinBlockSize: The minimum size, in KB, of text that every mapper processes.
- MaxBlockSize: Maximum size, in KB, of text that every mapper processes.
- KMSKeyARN: The ARN of KMS key used to encript environment variables. (Optional)
- MapperMemory: The memory of the mapper Lambda functions. The maximum text size to process by every Mapper is restricted by this amount of memory.
- ReducerMemory: The memory of the reduce Lambda functions.
- TimeOut: The elapsed time for a Lambda function to run before terminating it.
- ReducersNumber: Number of reducers to use

- User function in mapper and reducer code needs to be configured according to the data input.

The input and output data is stored in the azure blob storage. The MapReduce application is developed on python 3.8 platform. As soon as the data(CSV file) is inserted into the blob storage it triggers azure function. The azure function splits the file into small chunks and coverts it into a uft-8 format.

6 Evaluation

The implemented MapReduce application with azure serverless services reduces the execution time of MARLA architecture and it is approximately around the Hadoop MapReduce completion time. The graph shows the execution time of different sizes of input file. The figure 4 shows the duration for execution for different file sizes.

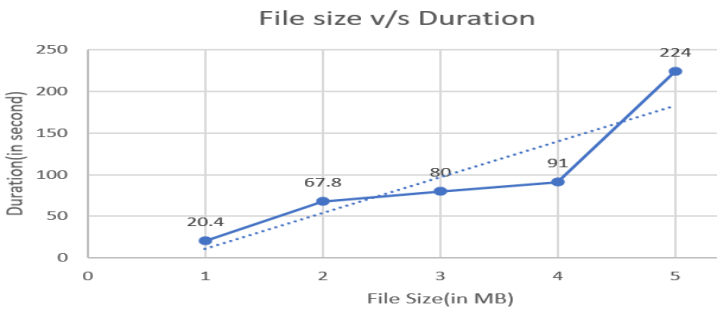

Figure 4: execution time for different file sizes

6.1 Experiment / Case Study 1

The file size of 2 MB is chosen to evaluate the performance by using parameters such as memory usage and cost of occurrence. A CSV test file has been created to execute azure function. After execution it creates a CSV file to display results. Since the execution of the test case is successful, the results are displayed in the CSV file and the execution details can be observed in the function insights.

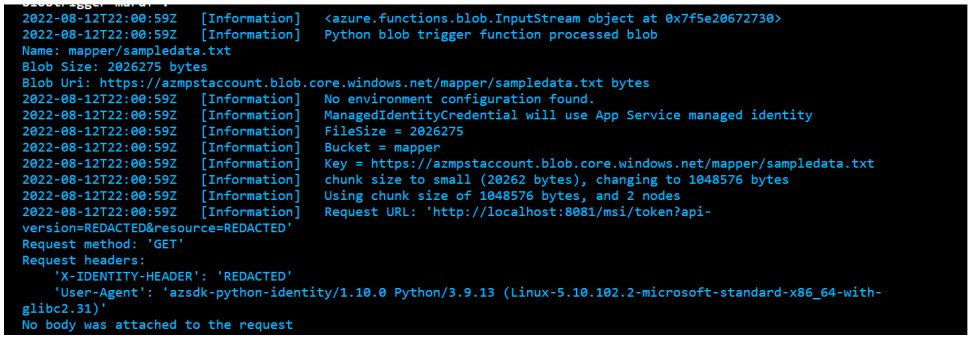

Figure 5: Executing 2 MB file

6.2 Experiment / Case Study 2

The file size of 8 MB is chosen to evaluate the performance by using parameters such as memory usage and cost of occurrence. A CSV test file has been created to execute azure function. After execution it creates a CSV file to display results. Since the execution of the test case is successful, the results are displayed in the CSV file and the execution details can be observed in the function insights.

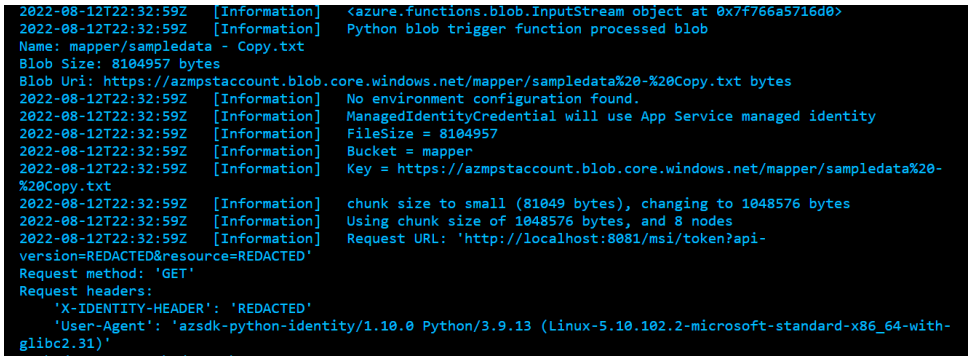

Figure 6: Executing 8 MB file

6.3 Discussion

The varying file size from 1MB to 5 MB are executed on the implemented application in which the overall completion time is 1.6 minutes for all sizes of data. The resultant latency of 12 ms(in Figure: 7 is due to the cold start problem.

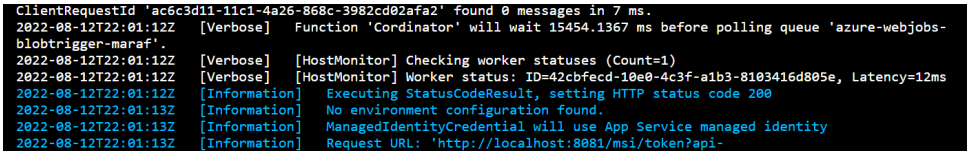

Figure 7: 12 ms of latency due to cold start

The Azure monitor provide the graphical representation of duration, error count, throttle time and invocation of the Azure Functions. The graph duration graph shows the time taken by the coordinator function to start the processing of the data when the input file is uploaded in the Azure blob storage. The Figure 8 shows the ratio of execution time required for the processing by different operations such as mapper, coordinator and reducers. It also shows the number of invocation involved for these operations. The Figure 9 shows the CPU utilization by the different invocations and the duration of each request for processing the data.

OPERATION NAME	DELTA	DURATION (AVG)	COUNT
Overall		**1.68 min**	**44**
Mapper	↗ 906.05%	2.75 min	25
Cordinator	↗ 31.32%	38.4 sec	7
Reducer	↘ -6.89%	2.37 sec	12

Figure 8: Function execution time for different operation

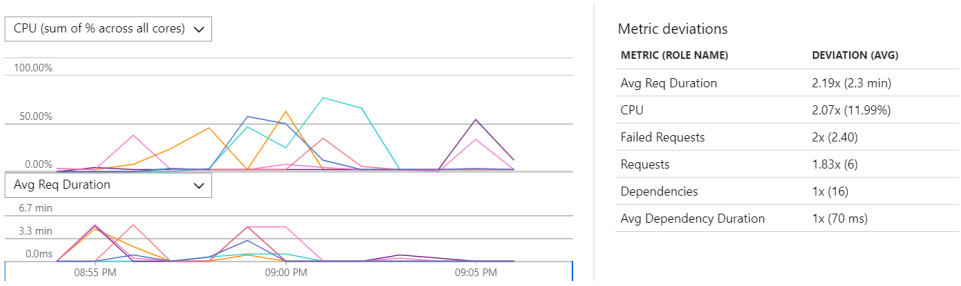

Figure 9: CPU utilization and request during the different size of input files

The below table gives an account of execution time required by different platforms for executing MapReduce in serverless infrastructure. The implemented mapreduce application takes almost 5x more time than MARLA application on AWS platform and additionally it is expensive to operate. The Figure 10 shows the comparison of the execution time required by MARLA on AWS and Azure for different file sizes. This paper shows that the Azure MARLA model require 5x more time to complete its execution of test data and the cost of execution also increases accordingly.

Platform	Time of execution(in sec)	Cost (in Dollars)
MARLA on Azure	96	0.0000433
MARLA on AWS	18	0.0000266
AWS EMR	28	0.00086
Hadoop MapReduce	183	$

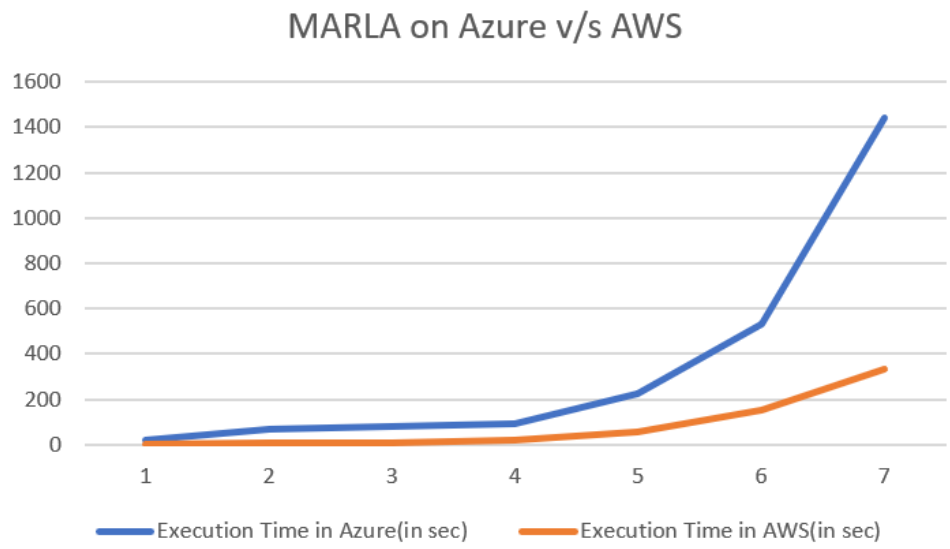

Figure 10: Execution time comparison of MARLA on Azure and AWS

7 Conclusion and Future Work

The presented paper has implemented MARLA application on Azure. According to the study carried out in the thesis, performance gaps can be reduced by introducing fine-grained flexibility in favor of cutting-edge function-as-a-service and backend-as-a-service technologies. The presented paper has used Azure functions and Azure blob storage effectively to give optimal performance. The Azure Function has uniform latency behavior when compared to other platforms which has no impact on the processing duration of MapReduce operations. To aim this, a deep analysis of CPU utilization and execution time of Azure functions is performed. A major focus is paid on error rate, invocations, throttle time and cost of execution. The research has following findings:

- MARLA architecture on AWS has almost 5x better makespan than MARLA on azure
- The difference between their makespan increases as the file size increases and it increase significantly after 5MB file size.

Due to restricted resources, the presented paper has used blob storage for storing intermediate data. For future scope, an analysis can be carried out to make use of cache

data storage for intermediate data that may enhance the overall turn around time of the application. Also, the presented paper has implemented the MapReduce application for small analytic platform which can be expanded for the larger analytical platforms.

References

Alotaibi, M. T., Almalag, M. S. and Werntz, K. (2020). Task scheduling in cloud computing environment using bumble bee mating algorithm, *2020 IEEE Global Conference on Artificial Intelligence and Internet of Things (GCAIoT)*, Dubai, United Arab Emirates, pp. 01–06. Core Ranking 2021: NA.

Alsaidy, S. A., Abbood, A. D. and Sahib, M. A. (2020). Heuristic initialization of pso task scheduling algorithm in cloud computing, *Journal of King Saud University-Computer and Information Sciences* . JCR Impact Factor 2021: 13.473.

Bhattacharya, S. and González-Vélez, H. (2020). 3d-stacked memory for shared-memory multithreaded workloads, *in* M. Steglich, C. Mueller, G. Neumann and M. Walther (eds), *Proceedings of the 34th International ECMS Conference on Modelling and Simulation, ECMS 2020*, European Council for Modeling and Simulation, Wildau, pp. 368–375. CORE2021 Rank: B.
URL: *https://doi.org/10.7148/2020-0368*

Burkat, I., Pawlik, M., Balis, B., Malawski, M., Vahi, K., Rynge, M., da Silva, R. F. and Deelman, E. (2021). Serverless containers – rising viable approach to scientific workflows, *2021 IEEE 17th International Conference on eScience (eScience)*, pp. 40–49.

Dai, T. and Fan, X. (2021). Multi-stove scheduling for sustainable on-demand food delivery, *Sustainability* **13**(23): 13133.

Djemame, K., Parker, M. and Datsev, D. (2020). Open-source serverless architectures: an evaluation of apache openwhisk, *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, Leicester, UK, pp. 329–335. Core Ranking 2021: unRanked.

D'Mello, G. and González-Vélez, H. (2019). Distributed software dependency management using blockchain, *PDP 2019: 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, IEEE, Pavia, pp. 132–139. CORE2021 Rank: C.
URL: *https://doi.org/10.1109/EMPDP.2019.8671614*

Dong, J., Goebel, R., Hu, J., Lin, G. and Su, B. (2021). Minimizing total job completion time in mapreduce scheduling, *Computers and Industrial Engineering* **158**: 107387. JCR Impact Factor 2021: 5.431.
URL: *https://www.sciencedirect.com/science/article/pii/S0360835221002916*

Elbeltagi, E., Hegazy, T. and Grierson, D. (2005). Comparison among five evolutionary-based optimization algorithms, *Advanced engineering informatics* **19**(1): 43–53. JCR Impact Factor 2021: 5.603.

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**: 259–274. JCR Impact Factor 2021: 7.187.

Gu, R., Yang, X., Yan, J., Sun, Y., Wang, B., Yuan, C. and Huang, Y. (2014). Shadoop: Improving mapreduce performance by optimizing job execution mechanism in hadoop clusters, *Journal of Parallel and Distributed Computing* **74**(3): 2166–2179. JCR Impact Factor 2021: 3.737.
URL: *https://www.sciencedirect.com/science/article/pii/S0743731513002141*

Gunasekaran, J. R., Thinakaran, P., Nachiappan, N. C., Kandemir, M. T. and Das, C. R. (2020). Fifer: Tackling resource underutilization in the serverless era, *Proceedings of the 21st International Middleware Conference*, Middleware '20, Association for Computing Machinery, New York, NY, USA, p. 280–295. Core Ranking 2021: A.
URL: *https://doi.org/10.1145/3423211.3425683*

Hirtan, L., Dobre, C. and González-Vélez, H. (2020). Blockchain-based reputation for intelligent transportation systems, *Sensors* **20**(3): 791. JCR Impact Factor 2021: 3.576.
URL: *https://doi.org/10.3390/s20030791*

Houssein, E. H., Gad, A. G., Wazery, Y. M. and Suganthan, P. N. (2021). Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends, *Swarm and Evolutionary Computation* **62**: 100841. JCR Impact Factor 2021: 7.177.

Huang, T.-C., Huang, G.-H. and Tsai, M.-F. (2022). Improving the performance of mapreduce for small-scale cloud processes using a dynamic task adjustment mechanism, *Mathematics* **10**(10): 1736.

Kalavri, V. and Vlassov, V. (2013). Mapreduce: Limitations, optimizations and open issues, *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, NW Washington, DC,United States, pp. 1031–1038. Core Ranking 2021: B.

Khedekar, V. and Tian, Y. (2020). Multi-tenant big data analytics on aws cloud platform, *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0647–0653. JCR Impact Factor 2021: NA.

Koschel, A., Klassen, S., Jdiya, K., Schaaf, M. and Astrova, I. (2021). Cloud computing: Serverless, *2021 12th International Conference on Information, Intelligence, Systems Applications (IISA)*, pp. 1–7. JCR Impact Factor 2021: NA.

Kruekaew, B. and Kimpan, W. (2022). Multi-objective task scheduling optimization for load balancing in cloud computing environment using hybrid artificial bee colony algorithm with reinforcement learning, *IEEE Access* **10**: 17803–17818. JCR Impact Factor 2021: 3.367.

Leal, F., Veloso, B., Malheiro, B., González-Vélez, H. and Burguillo, J. (2020). A 2020 perspective on "scalable modelling and recommendation using wiki-based crowdsourced repositories:" fairness, scalability, and real-time recommendation, *Electronic Commerce Research and Applications* **40**: 100951. JCR Impact Factor 2021: 6.014.
URL: *https://doi.org/10.1016/j.elerap.2020.100951*

Lee, H., Satyam, K. and Fox, G. (2018). Evaluation of production serverless computing environments, *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, San Francisco, CA, USA, pp. 442–450. Core Ranking 2021: B.

Louboutin, M. (2020). *Modeling for inversion in exploration geophysics*, PhD thesis, Georgia Institute of Technology.

Moseley, B., Dasgupta, A., Kumar, R. and Sarlós, T. (2011). On scheduling in map-reduce and flow-shops, *SPAA '11: Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, SPAA '11, Association for Computing Machinery, New York, NY, USA. Core Ranking 2021: A.
URL: *https://doi.org/10.1145/1989493.1989540*

Muchhala, Y., Singhania, H., Sheth, S. and Devadkar, K. (2021). Enabling mapreduce based parallel computation in smart contracts, *2021 6th International Conference on Inventive Computation Technologies (ICICT)*, pp. 537–543. JCR Impact Factor 2021: NA.

Nikitas, N., Konstantinou, I., Kalogeraki, V. and Koziris, N. (2021). Cherry: A distributed task-aware shuffle service for serverless analytics, *2021 IEEE International Conference on Big Data (Big Data)*, Orlando, FL, USA, pp. 120–130. Core Ranking 2021: B.

Saha, A. and Jindal, S. (2018). Emars: Efficient management and allocation of resources in serverless, *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, San Francisco, CA, USA, pp. 827–830. Core Ranking 2021: B.

Sampé, J., Vernik, G., Sánchez-Artigas, M. and García-López, P. (2018). Serverless data analytics in the ibm cloud, *Middleware '18: Proceedings of the 19th International Middleware Conference Industry*, Middleware '18, Association for Computing Machinery, New York, NY, USA, p. 1–8. Core Ranking 2021: A.
URL: *https://doi.org/10.1145/3284028.3284029*

Sandholm, T. and Lai, K. (2009). Mapreduce optimization using regulated dynamic prioritization, *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '09, Association for Computing Machinery, New York, NY, USA, p. 299–310. Core Ranking 2021: A*.
URL: *https://doi.org/10.1145/1555349.1555384*

Shafahi, Z. and Yari, A. (2021). An efficient task scheduling in cloud computing based on aco algorithm, *2021 12th International Conference on Information and Knowledge Technology (IKT)*, Babol,Iran, pp. 72–77. Core Ranking 2021: NA.

Shi, Y., Zhang, K., Cui, L., Liu, L., Zheng, Y., Zhang, S. and Yu, H. (2016). Mapreduce short jobs optimization based on resource reuse, *Microprocessors and Microsystems* **47**: 178–187. JCR Impact Factor 2021: 1.525.
URL: *https://www.sciencedirect.com/science/article/pii/S0141933116300436*

Weets, J., Kakhani, M. and Kumar, A. (2015). Limitations and challenges of hdfs and mapreduce, *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, IEEE Computer Society, Los Alamitos, CA, USA, pp. 545–549. JCR

Impact Factor 2021: NA.
URL: *https://doi.ieeecomputersociety.org/10.1109/ICGCIoT.2015.7380524*

Witte, P. A., Louboutin, M., Modzelewski, H., Jones, C., Selvage, J. and Herrmann, F. J. (2020). An event-driven approach to serverless seismic imaging in the cloud, *IEEE Transactions on Parallel and Distributed Systems* **31**(9): 2032–2049. JCR Impact Factor 2021: 2.687.

Xiong, Y., Peng, Q. and Zhang, Z. (2020). Research on mapreduce parallel optimization method based on improved k-means clustering algorithm, *Proceedings of the 3rd International Conference on Data Science and Information Technology*, DSIT 2020, Association for Computing Machinery, New York, NY, USA, p. 47–52. Core Ranking 2021: NA.
URL: *https://doi.org/10.1145/3414274.3414282*