National College of Ireland

Ant Colony Optimization for MapReduce Application to Optimise Task Scheduling in Serverless Platform

MSc Research Project
Cloud Computing

Shweta Das
Student ID: x20138547

School of Computing
National College of Ireland

Supervisor: Sean Heeney

Ant Colony Optimization for MapReduce Application to Optimise Task Scheduling in Serverless Platform

Shweta Das
x20138547

Abstract

Rising demand in computation power gained significant interest towards serverless computing. In serverless billing generates only for code execution. In serverless inefficient tasks, scheduling will result in over-provisioning or under-provisioning resources which will incur more cost. Therefore it is very important to schedule the task in serverless as it will impact the cost in serverless. In the proposed paper, the efficiency of the serverless platform is enhanced by implementing Ant Colony Optimization(ACO) with a map-reduce application. ACO is a paradigm of designing a meta-heuristic algorithm that shares a common approach to construct a solution provided by both standard construction and previously constructed solutions. Since map-reduce is best suitable for big data problems but it creates additional overhead. To overcome the additional map-reduce overhead Ant Colony Optimization(ACO) is used along with fast and slow cloud storage. In the proposed paper, the existing MARLA architecture is addressed and improvisation in MARLA architecture is elaborated with the help the of ACO algorithm. Further results were compared and observed with and without implementation the of ACO algorithm.

1 Introduction

Cloud technology is one of the emerging sectors in Information Technology. Cloud computing provides access to on-demand large computational power which can be easily scaled up and scale down as and when required . With the rise of cloud technology, various cloud vendors were also emerging in the market due to which users can enjoy the quality of service at a low cost. Many organizations are using the cloud due to no hardware upfront cost and its unique pay-as-you-go billing scheme. The usage of such technology gives rise to a large amount of daily data. To handle the massive data generated by cloud users requires a centralized data center that consists of storage devices. Managing data is one of the critical tasks in the cloud. Hence, efficient task scheduling techniques can provide better scalability and performance in the cloud.

Nowadays many IT organizations are utilizing cloud infrastructure for daily business transactions. Due to the increasing demand for cloud infrastructure cloud vendors are also expanding cloud capabilities and thus create more options for cloud users. Cloud vendors provide various configuration options to cloud users(small scale to large scale). The reliability of any cloud vendor is dependent upon the Quality of Service (QoS) and the cost of service . Cloud vendors' expenses are determined by the amount of electricity consumed by their servers. Sinc, cloud handles the number of large processes and if they are not planned properly then it might create loss to cloud vendor revenue. Improper process allocation can result in under-provisioning or over-provisioning of resources. Therefore it is very important to schedule tasks in the cloud to optimally use resources. It will help cloud vendors to enhance their Quality of Service(QoS) and result in minimizing the violation of the Service Level Agreement(SLA) optimize their service cost. Since platform as a service and cloud as a service both provide monitoring of the resources of every application therefore task scheduling is necessary.

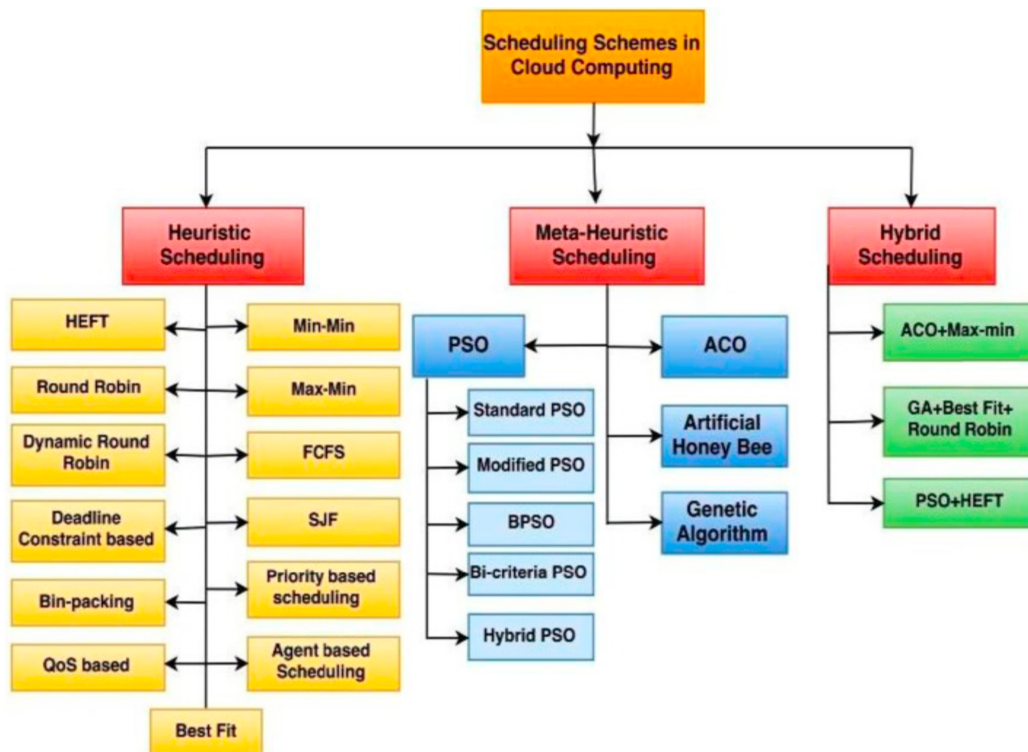

Figure 1: Classification of task scheduling algorithms Kumar et al. (2019)

If the number of clients was increased in the cloud then scheduling becomes a challenging task hence an appropriate scheduling algorithm needs to be implemented. In serverless, the key challenge of task scheduling is to schedule tasks for each class of service to maximize the revenue and meeting the user's needs. To integrate an efficient scheduling algorithm first we need to analyze the existing task scheduling algorithms their advantages and limitations. Task scheduling algorithms are of two types heuristic and meta-heuristic algorithms. Heuristic algorithms can be further classified into list-based, clustering-based and duplication-based. The motive of this paper is to identify and integrate the task scheduling algorithm in function-as-a-service(FAAS). In the proposed research we are focusing on Ant Colony Optimisation(ACO) of the Meta-Heuristic algorithm family. Ant Colony Optimisation is based on ants' behavior. Ants are governed by the goal of finding food. Ants are wandering about their colonies while searching. An ant hops from one location to another in search of food. It leaves an organic molecule known as pheromone on the ground as it moves. Pheromone trails are used by ants to communicate with one another. When an ant discovers some food, it transports as much as it can. When it returns, it leaves pheromones on the routes based on the amount and quality of the food. Ants can detect pheromones. As a result, other ants can smell it and follow it. The higher the pheromone level, the more likely that way will be chosen, and the more ants that follow the path, the more pheromone will be found on that trail.

1.1 Motivation

To efficiently utilize the resource in the serverless platform using meta-heuristic task scheduling algorithms such as Ant Colony Optimization(ACO). The main aim is to implement high latency shuffling of MARLA architecture in serverless platforms to achieve low latency and low cost of execution.

1.2 Research Question

The objective of the research work is to provide efficient task scheduling for map-reduce applications in the serverless cloud platform. Analyzing the benefits and constraints of using ACO scheduling algorithm in the serverless platform. Using two types of cloud storage(fast and slow) provides low-cost utilization for big data problems. As a result, this study project answers the following question:

Can the makespan of task scheduling in the MapReduce application be improved by using the Ant Colony Optimization algorithm(ACO) in Serverless platforms e.g. Amazon Web Services(AWS) or Openstack Qinling?

1.3 Report Structure

Section 2 expands on the related work in the field of task scheduling in cloud computing. It also covers the constraints and future work in task scheduling by many authors. Section 3 describes the methodology and pre-requisite required for implantation of ACO in Map Reduce application. Further, section 4 elaborates the MARLA architecture and our proposed model to improvise the task scheduling in the cloud. Section 5 and 6 shows the implantation of ACO in serverless platform and evaluation matrices based on different conditions. Lastly, section 7 describes the future work and conclusion of the proposed research work.

2 Related Work

This section elaborates various research-related work in serverless platform. This section is divided into four parts 2.1 review on task scheduling algorithms in cloud computing, 2.2 Review on serverless infrastructure, 2.3 review on performance analysis in serverless, 2.4 review on MapReduce in serverless infrastructure, 2.5 analysis of related work.

The Gu et al. (2014) presentation outlines current trends in serverless computing. According to the author, serverless has eliminated the need for developers to provision and manage servers. For developers to achieve desired business logic, they should focus more on functions than on business logic. The author outlines the drawbacks of using serverless for consumers and providers. FaaS models have some limitations for consumers including not providing users with the latest versions of libraries of software. Providing services involves managing the lifecycle of a user's function, scalability, and fault tolerance. The result will be that developers will be able to design applications that are focused on platform behavior. There are also cost, cold start, resource limits, security, scaling, hybrid cloud, legacy system challenges that the author discusses for provider and user. Besides minimizing the cost of resource usage when executing tasks and when it's idle, Serverless faces a fundamental challenge cost. To reduce costs, an optimal scaling technique needs to be addressed for the charge to zero during idle time. To prevent resource limitations, the platform should be managed during spikes in load and during attacks. As multi-user platforms run many functions, the provider should adapt strongly to isolate functions. As and when necessary, the provider should make sure that functions are elastic and scalable. Serverless presents some challenges for developers in terms of configuration, deployment, IDEs, concurrency, scalability, monitoring and debugging.

2.1 Review on Task Scheduling Algorithms in Cloud Computing

Particle Swarm Optimization algorithm(PSO), Genetic Algorithm(GA), Ant Colony Optimisation(ACO), and Cuckoo Search(CS) are heuristic algorithms used for resolving task scheduling problems. According to Al-maamari and Omara (2015), they developed a Dynamic Adaptive Particle Swarm Optimization algorithm (DAPSO) for fewer makespans and more resource availability. During an experiment, the author discovered that the execution speed of DAPSO on 10 VMs was superior to the default PSOCS and PSO algorithms by 44.29 percent and 23.75 percent, respectively. By using the DAPSO algorithm on five virtual machines, resource utilization was 36.19% and 20.60% better than the default PSO and PSOCS algorithms. According to the author, changing inertia weight and trapping on local search improves task scheduling and resource utilization.

An optimization strategy for task scheduling that uses ant colonies (ACO) is presented by author Wei (2020). In the paper, cloud-based scheduling models are used for improved ACO, thus avoiding optimization problems. A task scheduling function is then created by combining three objectives: minimum waiting time, degree of resource lot balance, and task completion cost. An experiment was conducted using Cloudsim, an open-source simulation tool. Tests were conducted on a Windows 10 machine with an Intel Core i5-7300HQ CPU and 4GB RAM. In experiments, 40,80,120,160,200 tasks with instruction sets ranging from 5000-5000000 are processed. Based on the simulation, it was found that the algorithm has the highest utilization rate, balanced load, shortest completion time, and fastest coverage.

Parallelization has resulted in a significant reduction in resource usage. Various researchers proposed algorithms to solve these problems so that the resource can be fully utilized. Wang et al. (2020), an optimized solution is presented that considers the deadline constraint to reduce the cost-make-span. Using task scheduling, resources are dynamically allocated and utilized to reduce execution time and cost. To solve the current scheduling problem in cloud computing, the researchers developed an enhanced immune-based particle swarm optimization algorithm (IMPSO). By utilizing immunological memory, affinity, antibody concentration, antigen, and many more, an algorithm can be derived. It is divided into five stages, namely, calculate the affinity, clone antibodies, mutate antibodies, update memory sets, and replace the worst solutions. IMPSO results in significant savings and more efficient resource utilization when compared with other competitive solutions. With the proposed algorithm, you can get the best results in a shorter amount of time.

Ari et al. (2017) proposes a multi-objective algorithm (MO-ACO) based on ant colony optimization (ACO) to optimize task scheduling in cloud computing environments. The objective is to minimize the task duration and total cost while balancing the system load. The CloudSim simulator was used to conduct some experiments, and the results were then compared to those of the ACO and Min-Min algorithms. A comparison of MO-ACO and ACO and Min-Min algorithms reveals that the cost incurred is comparable. Additionally, using the MO-ACO algorithm resulted in a makespan of 5% less than the ACO or Min-Min algorithms. It was found that the MO-ACO algorithm has a smaller computational complexity when the system load is greater.

Scheduling tasks is another essential aspect to reduce cost by lowering power consumption, which will boost profits by both vendors and users. Overall processing time can be impacted by optimal resource allocation. The cloud can be considered green computing if the energy consumption is minimized. Guo (2017) presented a bioinspired version of ant colony optimization(ACO) algorithm called Cloud ACO(CACO) that optimizes the load balancing between resources while reducing execution time. With time and resource constraints, task scheduling plays a vital role in managing serverless loads efficiently. In this experiment, homogeneous scenarios are used. During the experiment, four identical Intel Xeon Core servers with 8GB of RAM were used as the experimental data center. Randomly generated jobs from 20 to 80 were used in 20-piece series with instruction sets between 500 and 1500 and from 1500 to 2000. CACO outperformed ACO and GA when the Cache Compression Optimization scenario was run 10 times.

Navimipour (2015), proposed a new algorithm for scheduling tasks in a cloud platform based on a bee colony. According to the proposed algorithm, the number of bees counts as several solutions. The first step consists of generating a random population. Population iteration begins after initialization with employees, observers, and scouts. There are four phases of the algorithm during experimental simulation: (1) the initialization phase, (2) the employee phase, (3) the observer phase, and (4) the scouting phase. Simulations were performed with a CPU Core i5 to which 4GB of RAM and 4MB of cache memory were allocated. When comparing the proposed algorithm with several alternative algorithms, it was found to improve execution time, missed tasks, as well as waiting times.

Grid computing and distributed computing are two components of cloud computing. The objective of cloud computing is to provide users with IT services on demand. A cloud service's reliability and availability are critical for a good user experience. Based on a Load Balancer mutation and particle swarm optimization (LBMPSO) based scheduling system, Awad et al. Awad et al. (2015), presented a mathematical model for allocating re-

sources using the Load Balancer mutation. LBMPSO offers a set of accountability metrics including reliability, execution time, makespan, workload balancing between tasks and resources, execution cost, and round trip. In the proposed algorithm, there are five parts: (1) task submission, (2) task information, (3) LBMPSO, (4) resource information, and (5) task buffer. Task buffers are primarily used to collect user tasks. This information about a task defines how it should be executed. Resources in cloud computing are data centers, hosts, and virtual machines (VM's) that provide information about available resources. As each task is submitted, it will receive an allocation of resources. Rescheduling failed tasks occurs during the process of resource allocation through the LBMPSO. LBMPSO allows task scheduling errors to be corrected in real-time, which improves reliability. To determine if LBMPSO can save makespan, execution time, transmission cost, and round trip time, the author compared PSO, Random Algorithm, and The Longest Cloudlet to Faster Processor (LCFP) to LBMPSO.

IMPVACO, an efficient and intelligent algorithm developed by Duan and Yong (2016), will be improved upon to improve the basic ant colony optimization algorithm. Adaptive modification strategy for pheromones is altered by IMVPACO so that the solution can be optimized directly based on pheromone increment. Ants have also adapted their moment strategies in such a way that they are mobile and able to solve large-scale problems. When this algorithm is applied to the traveling salesman problem, first it initializes all parameters, then it gives the classification of cities, then m number of ants are added to n number of the cities, and finally, that list is added to the tabu list. Table Tabu should not be null when dynamic movement rules determine the next coming city. A path length is calculated in the last step after selecting the best choice, followed by a pheromone update and finally the iteration control. An iteration technique that employs boundary symmetric mutations for symmetric mutations is used. This technique improves the mutation efficiency and accuracy of iteration results.

Tawfeek et al. (2013) discuss cloud computing issues that are related to task scheduling. There have been various meta-heuristic algorithms proposed and implemented to solve the existing optimization problems. Adaptability is one quality that a good task scheduler should possess. It should be able to change its scheduling approach as the environment and task category(s) change. In the experiment, ACO's performance has been measured with different Alpha values, different beta values, different tmax values, and various Ant numbers to determine its relative strengths and weaknesses. For average makespan with several task sets, the author implements Ant colony optimization (ACO), First Come First Serve (FCFS) and Round Robin algorithms (RR). ACO algorithm takes less time as the quantity task increases as compared to FCFC and RR algorithms. Another advantage is that it uses positive feedback mechanisms, allows internal parallelization, and it can be extended in various ways.

As a result of the strong positive feedback of Ant Colony Optimisation(ACO), the author Liu et al. (2014) proposed genetic adaptation of ant colonies for optimum performance. The algorithm uses the global search capability of genetic algorithms to arrive at an optimal solution as quickly as possible, and later converts the original pheromone of ACO. A total of 100 ants were used for the experiment in GA, crossover rate 0.6, mutation rate 0.1. When 28 iterations were used in GA, the search rate for optimal tasks was 98%. The GA rate reaches 63% upon reaching 50%, and the ACO rate reaches 95% upon reaching 50%. When merging the GA and ACO, the author found that task scheduling in the cloud can be solved more successfully.

2.2 Review on Serverless Infrastructure

The author evaluates performance of concurrently invoked functions on serverless platforms, including Amazon Lambda, Google Cloud Function, Microsoft Azure Function, and IBM Cloud Function. In this study, Lee et al. (2018) found that Amazon Lambda offers better performance in various aspects, including CPU performance, network bandwidth, and file I/O throughput. Because serverless computing has zero delays while booting up a new instance, and it is charged based on function invocation, serverless computing is considered more efficient than traditional virtual machines. Three trigger types were introduced: (1) HTTP, (2) Database, and (3) Object Storage. Triggers are executed whenever users perform operations. There are multiple methods of content delivery with HTTP triggers, including JSON, text, and other types DELETE, POST and PUT. Database triggers are triggered when an entry/deletion/modification is made in the database. The test is performed on a serverless environment, considering the CPU's response time, the run-time overheads, and the performance of temporary directories. To analyze the performance of an instance, the author performs a concurrent function throughput. Multiple CPU-intensive workloads can be processed concurrently without using any other resources. There is consistency for 1 concurrent non-parallel invocation, but there is an overhead of 28 to 4606% if 100 simultaneous invocations are made. In the parallelism for disk-intensive workloads function, a temporary directory of up to 500MB is available that can be accessed by different applications such as tool storage, intermediate data storage, and extra libraries. With 100 invocations, a concurrency analysis of disk-intensive workload revealed Amazon and Google both generated significant execution overheads of 91%, 145% and 338%, respectively, while Microsoft failed to execute the function within the time limit (5 minutes). The performance of the network is significantly degraded due to heavy operations of file uploading and downloading in concurrent network-intensive workloads. As a result, such activities were distributed among several locations, where network integration was possible. Data transmission from Amazon S3 storage to Google object storage was four times faster with Amazon Lambda than with Google Functions from 1 to 100 concurrency levels. AWS Lambda triggers for HTTP have a throughput of 55.7 functions per second. Object storage triggers have a throughput of 25.16 functions per second, and AWS database triggers have 864.60 functions per second.

A resource management system is proposed for serverless cloud computing which focuses on optimizing the resource allocation among containers in the paper titled Efficient Management and Allocation of Resources in Serverless (EMARS) by Saha and Jindal (2018). The EMARS prototype was built on OpenLambda, which is an open-source platform for serverless computing. In comparison to AWS Lambda, OpenLambda functions have lower latency and memory utilization. It was tested on Ubuntu 16.04 with Python functions. It was found that if the memory was increased, latency could be reduced. A 6MB file had a latency of 3.3 seconds, whereas a 10MB file had a latency of 1.2 seconds.

To address the problem of under-utilization in serverless, the Gunasekaran et al. (2020) proposed fifer. FIFER is a framework for adaptive resource management, which intelligently handles function chains on a serverless platform. FFifer minimizes overall latency by proactively starting containers, an intelligent request batching system, container scaling, and container spawning to avoid cold starts. Fifer is composed of load balancing, function scheduling, bin packing with proactive scaling policies, and load balancing. Brigade was built on top of Fifer using 5KLOC of JavaScript. The Brigade

serverless platform is an open-source tool developed by Microsoft Azure. In the experiment, 80 compute cores were used, which increased the spawn rate to 80% and increased container use by 4x while improving cluster efficiency by 31%.

Locus is a serverless analytics platform presented by Pu et al. (2019). This paper emphasizes performance and cost-effectiveness. The strategy in this paper was to combine (1) low cost but slow data storage (2) high cost but fast data storage to achieve high performance while being cost-efficient. A small amount of fast storage is used to process data, thus improving performance and requiring slower storage for storing data. According to the author, Locus meets Apache Spark running time in CloudSort with small amounts of fast storage. The Locus system is found to be slower than Amazon Redshift. Still, Locus is a better option than Redshift. Using fast but expensive storage combined with slow but cheap computing, Locus was able to achieve performance like Apache Spark. By using it both performance and cost can be optimized.

2.3 Review on performance analysis in Serverless

An evaluation of serverless infrastructure in a low-latency, the high-availability environment was conducted by Bardsley et al. (2018). To determine the causes of latency, the experiment was conducted using both API Gateway and Lambda to Lambda calls triggered by Python 2.7 AWS Lambda. Two test runs were run based on the results of the first test run to ensure the requisite number of warm lambdas. According to the analysis of JMeter HTTP response time, there was a delay in the spike and a subsequent contention for its resources during the initial startup. The performance of the internal system components was evaluated using CloudWatch and X-Ray. The CloudWatch tool provides a set of metrics, graphs and indicates where in the application time is spent. By using cloud watch, it was observed that each function spent approximately 3ms processing requests from API Gateway and 55ms executing the Reverse Array Lambda. According to the author, there are some optimization strategies in serverless: (1) Requests can wait for a lambda to start before getting a resource; (2) Circuit breakers can eliminate unreliable system components to eliminate needless resources consumption.

McGrath and Brenner (2017) proposed a serverless prototype that emphasis on performance and throughput by comparing results from AWS Lambda, Google Cloud Functions, and IBM Apache OpenWhisk. The prototype program was developed in the .NET framework and deployed on Microsoft Azure. Because of its scalability and low latency, the author chose Azure Storage as the storage type. It consists of two parts: (1) a web service and (2) worker services. Worker services manage and execute containers via a REST API exposed by the platform's web service. The metadata of a triggered function is stored in the Azure Storage table, and its code gets stored in the Azure Storage Blob. Metadata for functions include Function identifiers, Language runtime data, Memory size, and a Code Blob URI. GUIDs (Global Unified Identifiers) are randomly generated during function creation. Language run times specify the language used for function code. At the moment, only Node.js functions are supported. An application's maximum memory consumption is determined by its memory size. Proposed prototypes support manual function invocation. A function is invoked by calling "/invoke". The communication between web services and workers is controlled by the message layer. There are cold queues and warm queues for each function. Cold queues indicate the amount of space available on the platform. Initially, the web service dequeues messages from the warm queue, and if no messages are found, it then dequeues messages from the cold queue.

The server will return an HTTP 503 service unavailable message if all workers have been fully allotted to running containers. A worker can be assigned to a specific function and be responsible for managing container allocation. Compared with other prototypes, the proposed prototype has better performance until 13 minutes, and beyond that point, significant latency occurs.

To reduce the operational costs of serverless, the Mahmoudi and Khazaei (2020) proposes an analytical performance model to optimize quality of service and resource utilization. Using AWS Lambda as an experimental study environment, the author validates how well the proposed model applies and is accurate. In this evaluation, the authors measured the average warm and cold response times, as well as the expiration threshold. Experimental analysis was performed using Python 3.6 on AWS lambda. Using a wide range of workloads for the analytical performance model, it shows 30% higher response times than the average warm-up response time. On CPU-intensive workloads, the warm default threshold is 2x warm and 2x cold default threshold is 2x cold. AWS's Fibonacci calculation shows 4211ms of warm default and 5961ms of cold default.

Lloyd et al. (2018) shows how serverless performance varies 15x based on the heater temperature, the VM and container temperature. Hosting a website can have a significant impact on infrastructure elasticity, load balancing, provisioning variety, infrastructure retention, and memory reservation size. AWS Lambda function was used by the author to perform random math calculations. There are multiple calculations in each array, each starting at a random location. It was discovered that low memory and CPU allocation caused Lambda functions deployed with large array sizes to fail. In an experiment, 9 stress levels from 2 to 9 were introduced with random number generators, and the time to execute the random function was observed to increase with the stress level. To service Microsoft Azure functions, a trigger function was written in C to log the App service instance id and current worker process id. VSTS (Visual Studio Team System) is used for generating the load and stress on functions. The cold runs load test was conducted and tested for 2 minutes. The warm runs were tested for 2 minutes, 5 minutes, and 10 minutes. The experiment observed a fourfold improvement in the performance of COLD when the function memory was increased from 128MB to 1536MB. COLD service shows performance degradation by 4.6 times when too many container requests are made to an individual host. Furthermore, serverless service performance begins to degrade after 10 minutes of execution for the functions which are executed first by 15x after 40 minutes of inactivity. The performance boost was 1.55x during WARM execution. VMs were observed to have an average uptime of 2 hours and 8 minutes. When memory size is small, Lambda is capable of allocating and retaining four times more containers to host microservices.

2.4 MapReduce in Serverless Infrastructure

With the increasing evolution of big data applications, there is a need to improve the job scheduling for MapReduce, which will improve MapReduce performance. By using instant message transfer to optimize MapReduce setup and cleanup work, Gu et al. (2014) proposes a reduction in time and cost to sensitive task scheduling and map reduction. Their implementation is a totally compatible optimized version of Hadoop called SHadoop. Through the comparing of SHadoop with standard Hadoop, it was determined that SHadoop shows significant improvement in performance by 25% than standard Hadoop for Word Count, Sort, Page Rank and KMean.

In the context of big data or Apache Hadoop, MapReduce is a model commonly used to analyze large amounts of data. Giménez-Alventosa et al. (2019) discusses AWS Lambda's suitability to support MapReduce jobs, and provides MARLA (MapReduce on Lambda), an open-source application for executing MapReduce jobs in Python on AWS Lambda. Secondly, a comprehensive performance analysis is conducted using many case studies. MapReduce jobs can be executed using MARLA on Lambda through a lightweight Python application. Lambda function groups are made up of three different groups: coordinator, mapper, reducer. This application uses two s3 buckets, one to store incoming data (input data) and the other to store output data that is collected during the post-processing phase. The MapReduce job's execution workflow will begin upon uploading data sets into the input s3. An s3 event is triggered by the event which in turn invokes the coordinator function. To determine the best size of the data partitions, the coordinator function splits the overall dataset into smaller chunks as specified (N), provided the mapper's function has enough ram to store such a large amount of data. Based on the performance comparison with AWS' serverless MapReduce architecture, it was found that lambda's performance heterogeneity strongly affects MapReduce jobs in terms of execution time. According to the study, AWS Lambda provides reliable computing performance for applications that use its constraints.

MapReduce is increasingly being used in both academic and industrial research fields to analyze and analyze big data sets. Kalavri and Vlassov (2013) discusses Hadoop framework limitations and compares the various MapReduce frameworks for open issues, future scopes, and trends. Despite its scalability, MapReduce has a relatively low processing capacity. MapReduce's performance depends on a variety of factors. The majority of MapReduce execution time is consumed by initializing, scheduling, and monitoring tasks. Due to the need for advanced programming skills, the implementation of MapReduce has programming model issues. Auto tuning and ease of use is another issue of MapReduce. The author discovered that various frameworks such as Hadoop++, ERAL, HAIL and many more were discovered that many open-source frameworks are restricting researchers from understanding and expanding them.

2.5 Summary of related work

Managing big data is becoming more and more complex as data grows rapidly. MapReduce plays an essential role in resolving the problem of big data. While using MapReduce, there is also an overhead - for mapping and for reducing. In Giménez-Alventosa et al. (2019), a MapReduce application is integrated with serverless (MARLA) to increase its performance. To accommodate the evolution of Serverless, and to allow users to be billed according to execution time, it is increasingly being adopted by organizations. In McGrath and Brenner (2017),Lloyd et al. (2018), Bardsley et al. (2018) and Mahmoudi and Khazaei (2020) the study evaluated how serverless platforms such as Amazon Lambda, Google Function, Azure Function, and OpenWhisk performed. AWS Lambda's performance is much better than other platforms in every research study. Considering the cost of running serverless, there has been researching done to optimize the serverless execution time so that the execution time is lowered, thus making serverless cheaper to run. In Wang et al. (2020), Awad et al. (2015), Al-maamari and Omara (2015), Navimipour (2015), Ari et al. (2017), Liu et al. (2014) and Guo (2017) emphasis on use of various task scheduling algorithms to optimise execution time and costs in cloud computing environments. From a research survey conducted by Liu et al. (2014), Ant Colony Optimisation(ACO)

shows that it has greater performance than other scheduling algorithms. As a result, in this proposed research, I am using Ant Colony Optimization (ACO) algorithm with Map Reducer in AWS Lambda.

3 Methodology

Firstly this paper tries to implement the Map Reducer in OpenStack but due to computing node limitation, it introduces large latency before executing the task. Therefore, extending MARLA architecture on Amazon Web Service(AWS) was performed. Though AWS has a service dedicated to AWS Elastic MapReduce (AWS EMR) it doesn't have any console management. In the paper, various other AWS cloud services such as AWS S3, AWS Lambda, Redis cache storage are used to provide flexible service. Amazon S3 acts as an event trigger for the map and reducer function whenever there is input data. Additionally, AWS CloudWatch provides the logs such as execution time, CPU utilization, memory allocation and many more.

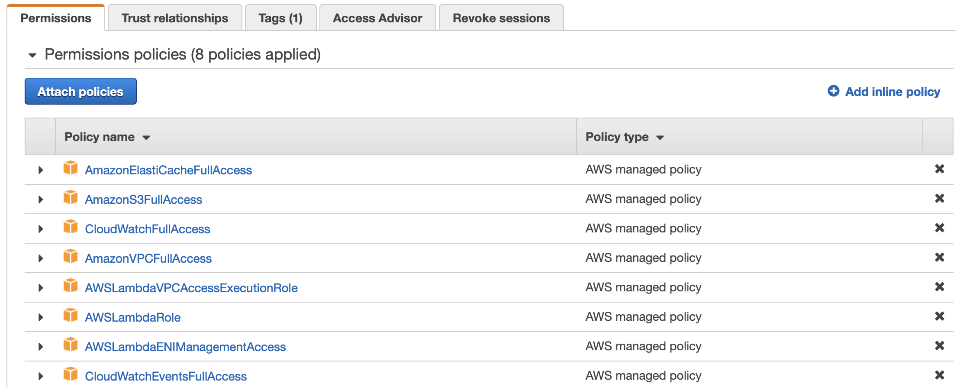

Figure 2: Policy attached in IAM

3.1 Material and Equipment's

Using Amazon Web Service(AWS), this paper provides fine-grained elasticity to existing MARLA architecture. As shown in figure 2, depicts different policies and roles required before creating lambda function. The AWS Lambda is used to execute the Map and Reducer function. Amazon S3 bucket access is provided to store and retrieve the stored data. AWS CloudWatch access is provided to monitor the performance of Lambda function in case of event failure it will trigger the alarm. Lastly, Amazon VPC is used to provide isolated infrastructure to map reducer applications which can be accessed through internet gateway.

3.2 Sample Data

The sample CSV data file has a size of 5MB and contains 12010 rows and 310 columns and is used to test the performance of the proposed architecture.

4 Design Specification

This section describes the design specification of the proposed research work. Sub-section 4.1, elaborates the existing system problem. In sub-section 4.2, the proposed system architecture flow is explained.

4.1 Existing system

As shown in figure 3, the existing MARLA architecture utilizes two AWS S3 buckets one for the incoming data stream and the other for processed data. Since map-reduce applications require high computational power and are time-consuming. In the existing system, there is no high-performance storage due to which it has latency issues.

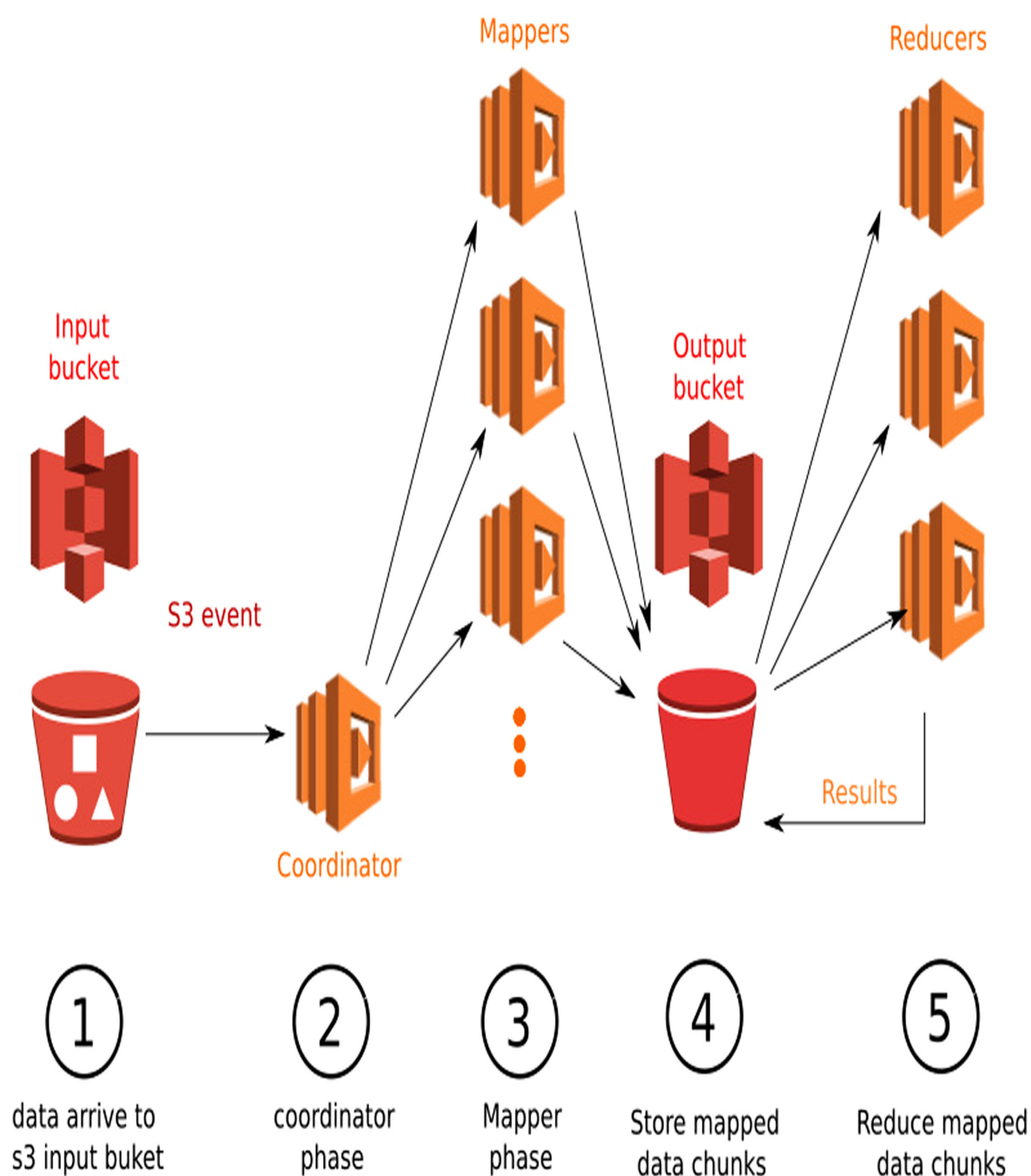

Figure 3: MARLA System Architecture Giménez-Alventosa et al. (2019)

4.2 Proposed System

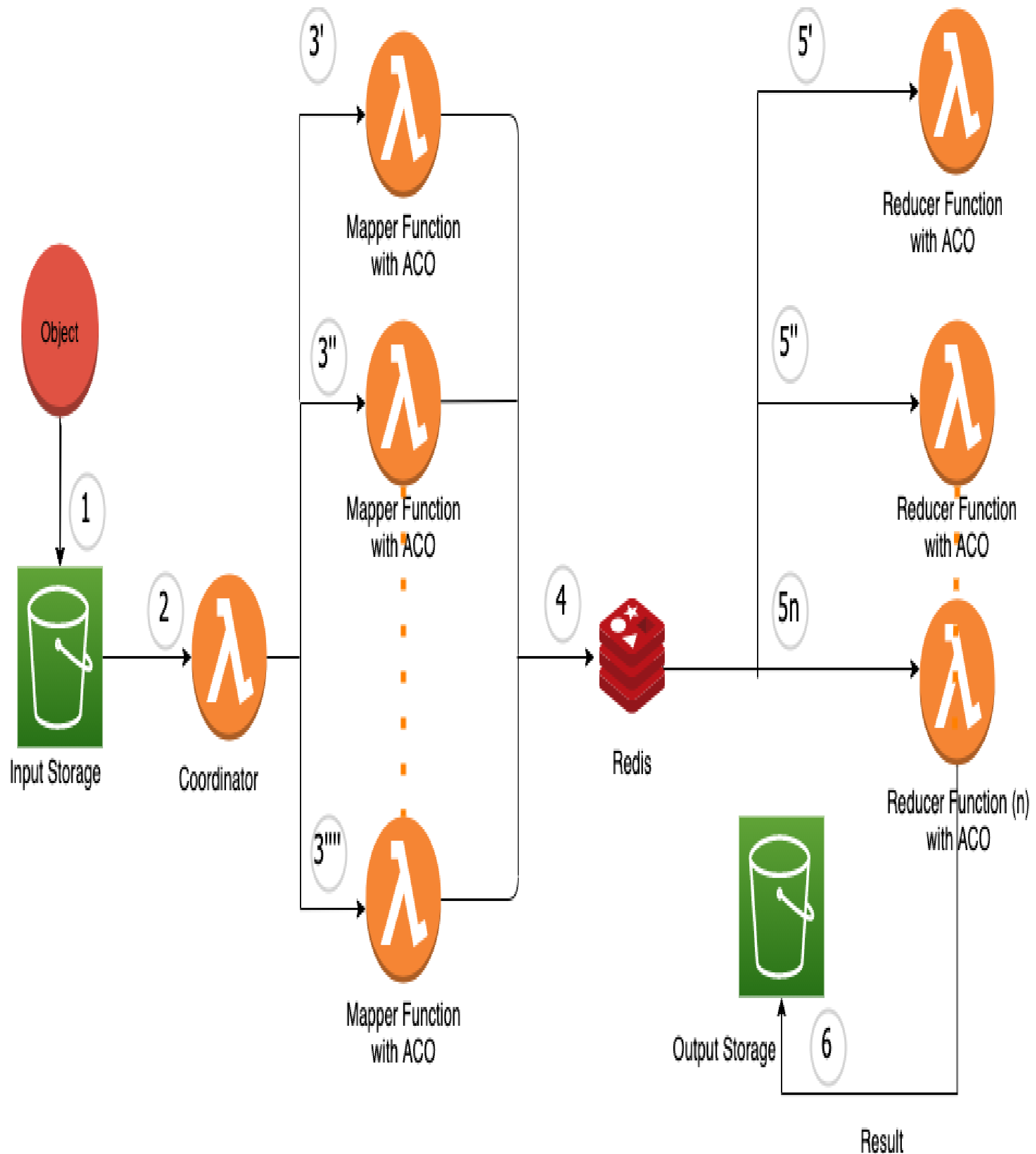

Figure 4: Proposed Research Architecture

Figure ?? depicts the proposed system architecture. The proposed system consists of two AWS S3 buckets. One bucket is for input object (CSV file) and another bucket is final output storage. We have used Redis as a temporary memory for handling data during the transition between the mapper and reducer function. The proposed system consists of Coordinator, Mapper and Reducer function. The coordinator function divides the data into equal chunks. The mapper function maps the data by assigning key-value pairs. Lastly, the reducer function performs complex computation on mapped data. With the help of the coordinator function, big complex problems were divided into small

chunks to boost the process of complex computation. The composite architecture is kept inside AWS VPC using AWS Direct Connect Link which enables MapReduce access in the proposed architecture. In the proposed architecture the AWS VPC is made up of different components such as NAT Gateway, Internet Gateway, Routing table and subnet. The architecture is mainly divided into three phases read, write and compute. The phases are as follows:

- **Compute:** For performing mapper and reducer functions, a lambda function is used which will reduce the assigned task. The provided lambda function is connected to S3 for input and final data and Redis for intermediate data which will be required during the shuffling phase.
- **Read:** The data is stored in AWS S3 which is further used to store the output data. AWS S3 provides easy restoring of every data version. The intermediate data generated by the mapper function is stored in Redis. Further, the intermediate data is pushed to the reducer function.

As shown in figure 4, the proposed architecture has 6 phases as:

- **Input Storage:** On uploading input data files such as CSV or text file into AWS S3 which will automatically trigger the coordinator function.
- **Coordinator Function:** The coordinator function divides the data into equal chunks to ensure all data is utilized in the map phase. On successful completion of the coordinator function, the mapper function is triggered.
- **Mapper Function:** The mapper function will map and assign the key-value pair to data chunks in 2D tuples such as Map[n][m].
- **Shuffling:** The mapped data is sorted and stored in Redis for further use in the reducer function.
- **Reducer Function:** In the reducer phase, the mapped key-value pairs were reduced with the help of 2D tuples.
- **Output Storage:** This phase stores the data processed by the reducer function. Each key is accompanied by the number of occurrences for each value.

5 Implementation

This section elaborates on the implementation of the proposed research work. The proposed design is implemented on AWS cloud and uses AWS S3, AWS Lambda and AWS Redis. We have implemented the MapReduce application on serverless for which we have created lambda function for mapper, coordinator and reducer. To implement Ant Colony Optimization(ACO) Algorithm with MapReduce application following configuration should be performed:

- Create an IAM role by assigning multiple policies to enable access to multiple services such as Cloud Watch, Lambda and S3 as shown in 2
- Create an IPv4 CIDR block of 192.168.0.0/16, 192.168.10.0/24, and 192.168.20.0/24 in a VPC to enable access to the internet using public subnet.

```python
import json
import boto3
import math
import redis

redis = redis.Redis(host='thesis.8sn3lf.0001.use1.cache.amazonaws.com', port=6379, charset="utf-8", decode_responses=True,)
s3 = boto3.client('s3')

def lambda_handler(event, context):
    # bucket = event['Records'][0]['s3']['bucket']['name']          // if dynamic allocation
    # key = event['Records'][0]['s3']['object']['key']              // if dynamic searching
    bucket = "thesis-test01"
    key = "grades.csv"

    response = s3.head_object(Bucket=bucket, Key=key)
    fileSize = response['ContentLength']
    fileSize = fileSize / 1048576
    print("FileSize = " + str(fileSize) + " MB")
    obj = s3.get_object(Bucket= bucket, Key=key)
    file_content = obj["Body"].read().decode("utf-8")

    #Calculate the chunk size
    chunkSize = ''
    MAPPERNUMBER=2
    MINBLOCKSIZE= 512
    chunkSize = int(fileSize/MAPPERNUMBER)
    numberMappers = MAPPERNUMBER
    if chunkSize < MINBLOCKSIZE:
        print("chunk size to small (" + str(chunkSize) + " bytes), changing to " + str(MINBLOCKSIZE) + " bytes")
        chunkSize = MINBLOCKSIZE
        numberMappers = int(fileSize/chunkSize)+1
    residualData = fileSize - (MAPPERNUMBER - 1)*chunkSize
```

Figure 5: MapReduce with ACO algorithm code snippet

- Create the security group which will allow incoming traffic and outgoing traffic. The inbound rule of this security group will be to allow traffic from anywhere to Redis, allow traffic from HTTP, HTTPS, SSH and custom TCP.
- Create an S3 bucket with allows public access and VPC endpoint to establish a private connection between VPC and S3 without any internet connectivity.
- Lastly, create the elastic cache for Redis so the data can be stored in Redis on a public and private network.

The input data and final output data in this paper are stored in Amazon S3 buckets. As shown in figure 5, the code snippet for MapReduce with ACO algorithm in AWS Lambda function is used for computing and Redis acts as a middleware temporary storage. The algorithm was written in python 3.8 and utilizes Boto3 to access AWS S3 bucket, Redis service, CSV library to import the CSV file, JSON to convert the input data structure into JSON format and math function for performing various mathematical Equations. Whenever there is data input inside the AWS S3 bucket(CSV file) it will trigger the lambda function. The lambda function will split the file into small chunks and convert it into a utf-8 format.

6 Evaluation

In the proposed research, the map-reduce application with Ant Colony Optimization(ACO) scheduling algorithm shows the nearby same performance as AWS EMR. Using fast(Redis) and slow(AWS S3) storage strategy the performance and execution time reduced as compare to existing MARLA architecture. The graph **??** depicts the execution time of MapReduce application for different file sizes such as 1MB, 2MB, 3MB, 4MB and 5MB respectively.

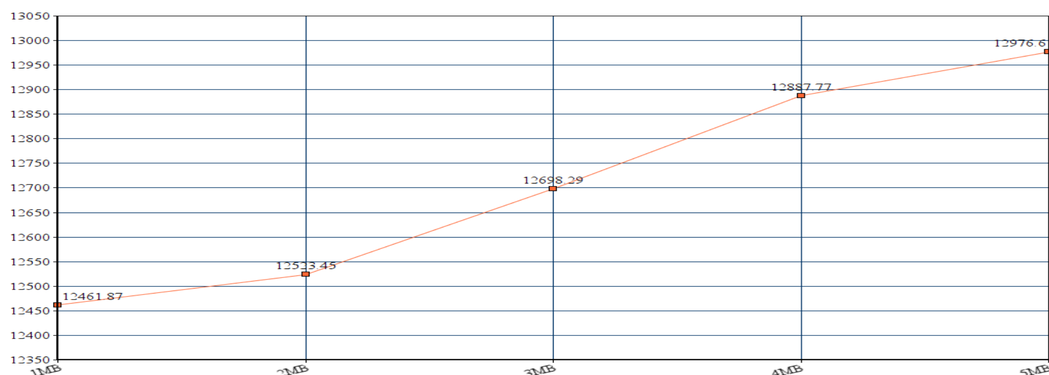

Figure 6: File size vs execution time

6.1 Case Study 1: Without ACO algorithm

As shown in figure 8, the successful execution for 1MB CSV file. The CSV file of 1MB is used to validate the memory usage and cost associated with the lambda function. For executing the lambda function we had created a JSON test file. After execution, it will display the result in JSON format. Since the test case for the 1MB file was successful hence its output is displayed in green colour. The execution result includes details such as billed duration, memory utilization and time taken for function execution. It can be seen that execution time was 12484.70ms and we billed for 12500ms, maximum memory utilization was 96MB and 704MB of resource were configured.

Figure 7: The Output for executing 1MB CSV file without ACO algorithm

6.2 Case Study 2: With ACO algorithm

As shown in figure 8, the successful execution for MapReduce application with ACO algorithm. The CSV file of 1MB is used to MapReduce application to check memory utilization and cost associated with a lambda function. Since the test case for the 1MB file was successful hence its output is displayed in green colour. The execution result includes details such as billed duration, memory utilization and time taken for function execution. It can be seen that execution time was 812.68ms and we billed for 900ms, maximum memory utilization was 95MB and 832MB of resources were configured.

Figure 8: The Output for executing 1MB CSV file with ACO algorithm

6.3 Case Study 3 (Unsuccessful)

As shown in figure 9, the execution of lambda function fails. This happens due to the large chunk size specified in code i.e. 2048 bytes. Here the code will divide the CSV file into 2048 bytes chunks and for processing the chunks we have allocated only 2 reducers hence it took some time which resulted in task timeout for the Lambda function.

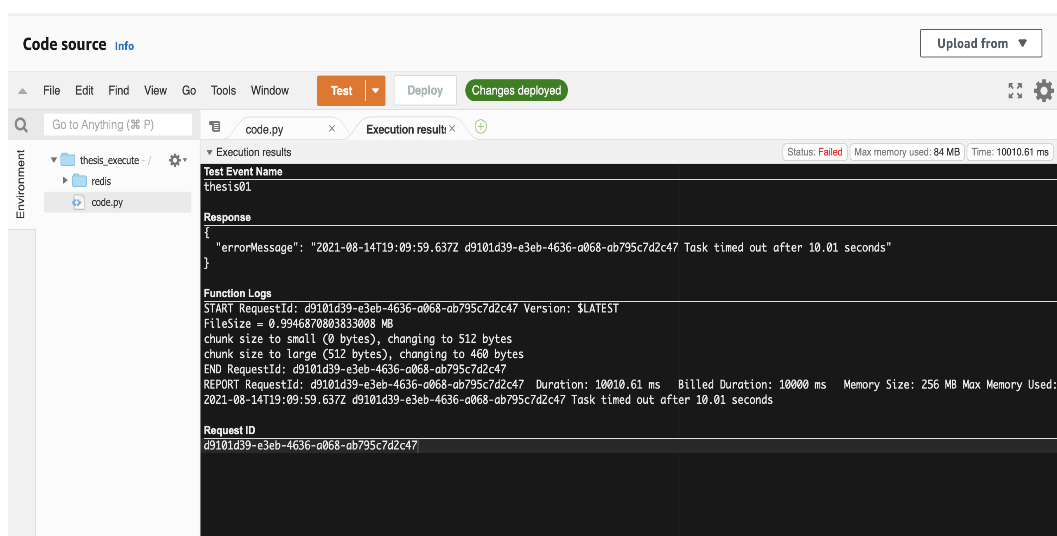

Figure 9: The output for 2048 bytes chunk size execution over 2 reducer

Redis is placed inside a virtual private cloud, hence the 10.67 second latency seen here. Since lambda utilizes resources from VPC hence it will result in an initial cold start. As shown below, the Amazon CloudWatch service provides graphs of error counts, invocation times, and duration. The graph 10, illustrates the error percentage for failed and successful executions. The green dot is 100 percent succeeded and the red dot is failed execution.

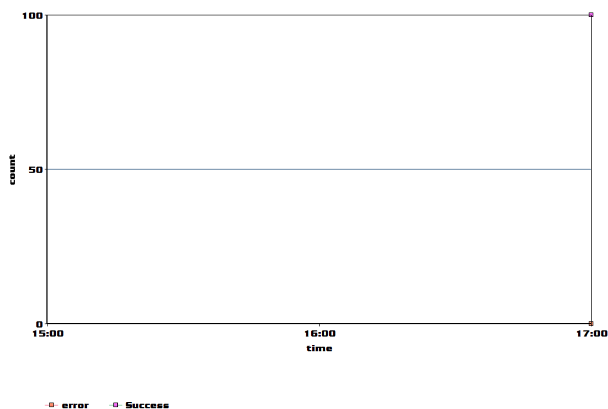

Figure 10: Error count for 1MB file size

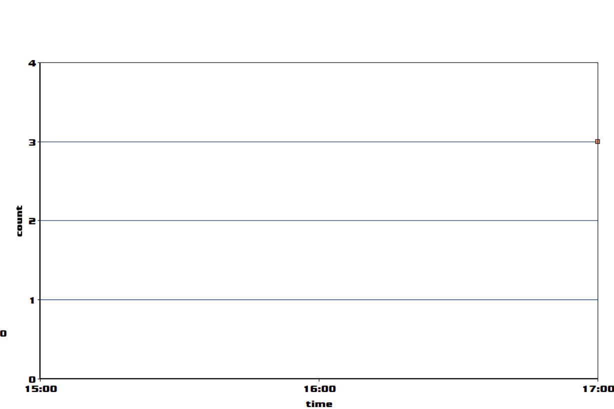

Figure 11: Invocation time for 1MB file size

The graph 11, depicts the function invocation count (number of time a function is called). Here lambda function is invoked two times first by AWS S3 event trigger and second while responding to the mapper function. The graph 12, shows the duration of the coordinator function from its invocation to stop invoking another function.

Figure 12: Function execution time for 5MB file size

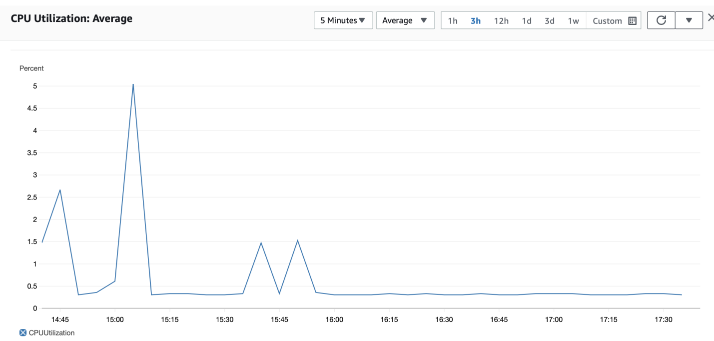

Figure 13: CPU utilization of Redis

The graph 13 depicts the CPU utilization by the Redis database. Here default threshold value has been set to 80% and it can be seen that the Redis threshold is under 45%. Figure 14 When extending Redis with Redis storage, this shows the replication factor. The graph below replication shows bytes as the amount of memory sent across all replicas by the master. In the proposed work we have used three Redis Elastic Cache replicas.

The graph 15 depicts the Redis network bytes inbound outbound during in-memory computation. It can be seen that network packet inbound is equal to network packet outbound (100% utilization).

The figure 16 depicts the cost incurred during the execution of the MapReduce application on AWS Lambda. The cost incurred by Elastic Cache was $26.37, Elastic Cloud

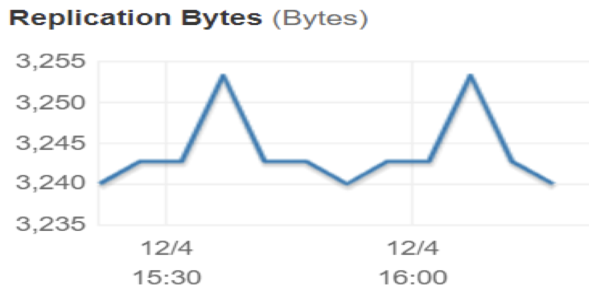

Figure 14: Replication graph of Redis

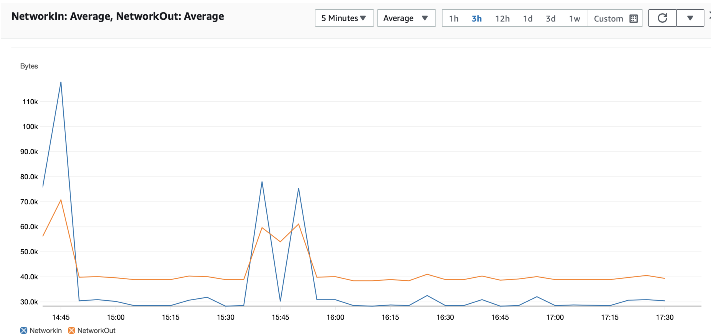

Figure 15: Redis network packets in and out

computing was $15.35 and the total cost incurred was $49.23.

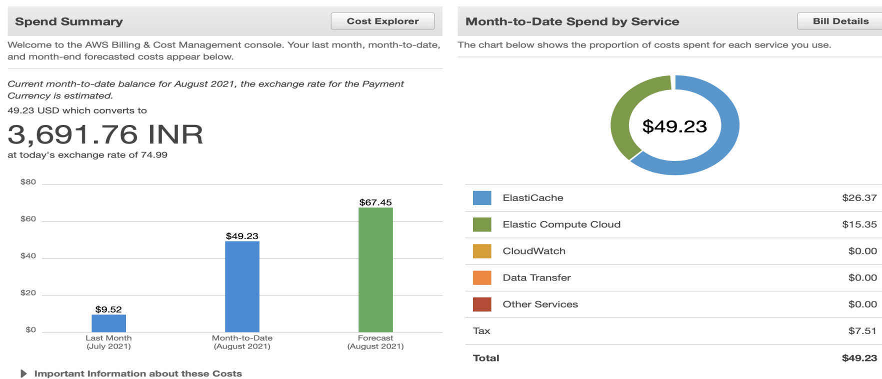

Figure 16: AWS Lambda billing for MapReduce application

7 Conclusion and Future Work

The paper aims at facilitating MapReduce on Amazon Web Services' serverless infrastructure via Ant Colony Optimisation(ACO) scheduling algorithm. The paper believes that proper task scheduling can facilitate the current lack of big data application support on serverless infrastructure. In the proposed research data transmission rate between resources is improved using Amazon S3 and Amazon Elastic Cache. On comparing the

results from other platforms, AWS Lambda demonstrates uniform latency behavior that does not influence the processing time of MapReduce processes. The results gained a significant deep analysis of CPU utilization of AWS Elastic Cache and duration, error, throttle and invocation of AWS Lambda. To this aim, the research focuses on:

- As a result of introducing AWS Elastic Cache, execution speeds were 4.5 times faster than with existing MARLA architecture.
- Due to the usage of cache memory the cost has also increased 2.5x times that of MARLA architecture.

Due to resource restrictions provided by AWS SLA, this proposed researchable to analyze small analytic performance. The future work comprises of expanding the task scheduling more efficiently using other task scheduling algorithms. Various hot and cold storage devices such as Redis, in-memory data store and many more storage combinations can be used to speed up the execution of big data applications in serverless.

References

Al-maamari, A. and Omara, F. (2015). Task scheduling using pso algorithm in cloud computing environments, *International Journal of Grid and Distributed Computing* **8**: 245–256.

Ari, A. A. A., Damakoa, I., Titouna, C., Labraoui, N. and Gueroui, A. (2017). Efficient and scalable aco-based task scheduling for green cloud computing environment, *2017 IEEE International Conference on Smart Cloud (SmartCloud)*, New York, NY, USA, pp. 66–71.

Awad, A., El-Hefnawy, N. and kader, H. A. (2015). Enhanced particle swarm optimization for task scheduling in cloud computing environments, *Procedia Computer Science* **65**: 920–929. International Conference on Communications, management, and Information technology (ICCMIT'2015).
URL: *https://www.sciencedirect.com/science/article/pii/S187705091502894X*

Bardsley, D., Ryan, L. and Howard, J. (2018). Serverless performance and optimization strategies, *2018 IEEE International Conference on Smart Cloud (SmartCloud)*, New York, NY, USA, pp. 19–26.

Duan, P. and Yong, A. (2016). Research on an improved ant colony optimization algorithm and its application, *International Journal of Hybrid Information Technology* **9**(4): 223–234.
URL: *https://gvpress.com/journals/IJHIT/vol9*$_n$*o4/20.pdf*

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. CoreRank: A, JCR Impact Factor: 6.125.
URL: *https://www.sciencedirect.com/science/article/pii/S0167739X18325172*

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. CoreRank:

A, JCR Impact Factor: 2.296.
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, USA, p. 280–295. CoreRank: A.
URL: *https://doi.org/10.1145/3423211.3425683*

Guo, Q. (2017). Task scheduling based on ant colony optimization in cloud environment, *AIP Conference Proceedings* **1834**(1): 040039.
URL: *https://aip.scitation.org/doi/abs/10.1063/1.4981635*

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*, Melbourne, VIC, Australia, pp. 1031–1038.

Kumar, M., Sharma, S., Goel, A. and Singh, S. (2019). A comprehensive survey for scheduling techniques in cloud computing, *Journal of Network and Computer Applications* **143**: 1–33.
URL: *https://www.sciencedirect.com/science/article/pii/S1084804519302036*

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, USA, pp. 442–450. CoreRank: B.

Liu, C., Zou, C. and Wu, P. (2014). A task scheduling algorithm based on genetic algorithm and ant colony optimization in cloud computing, *2014 13th International Symposium on Distributed Computing and Applications to Business, Engineering and Science*, Xi'an, China, pp. 68–72.

Lloyd, W., Ramesh, S., Chinthalapati, S., Ly, L. and Pallickara, S. (2018). Serverless computing: An investigation of factors influencing microservice performance, *2018 IEEE International Conference on Cloud Engineering (IC2E)*, Orlando, FL, USA, pp. 159–169.

Mahmoudi, N. and Khazaei, H. (2020). Performance modeling of serverless computing platforms, *IEEE Transactions on Cloud Computing* pp. 1–1. JCR Impact Factor=4.714.

McGrath, G. and Brenner, P. R. (2017). Serverless computing: Design, implementation, and performance, *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, Atlanta, GA, USA, pp. 405–410.

Navimipour, N. J. (2015). Task scheduling in the cloud environments based on an artificial bee colony algorithm, *Proceedings of 2015 International Conference on Image Processing, Production and Computer Science (ICIPCS'2015)*, Istanbul, Turkey, pp. 38–44.
URL: *http://uruae.urst.org/siteadmin/upload/1716U0615008.pdf*

Pu, Q., Venkataraman, S. and Stoica, I. (2019). Shuffling, fast and slow: Scalable analytics on serverless infrastructure, *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation*, NSDI'19, USENIX Association, Berkeley, USA, p. 193–206.

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

Tawfeek, M. A., El-Sisi, A., Keshk, A. E. and Torkey, F. A. (2013). Cloud task scheduling based on ant colony optimization, *2013 8th International conference on computer engineering & systems (ICCES)*, IEEE, pp. 64–69. address: Cairo, Egypt.

Wang, P., Lei, Y., Agbedanu, P. R. and Zhang, Z. (2020). Makespan-driven workflow scheduling in clouds using immune-based pso algorithm, *IEEE Access* **8**: 29281–29290. JCR Impact Factor: 3.745.

Wei, X. (2020). Task scheduling optimization strategy using improved ant colony optimization algorithm in cloud computing, *Journal of Ambient Intelligence and Humanized Computing* . JCR Impact Factor: 4.594.
URL: *https://doi.org/10.1007/s12652-020-02614-7*