National College of Ireland

A Comparative Analysis of Metaheuristic Algorithms for Optimizing Tasks in Serverless Frameworks for MapReduce Applications

MSc Research Project
Cloud Computing

Vaishnavi Sarjerao Waghmare
Student ID: 21172846

School of Computing
National College of Ireland

Supervisor: Prof. Shivani Jaswal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Vaishnavi Sarjearo Waghmare
Student ID:	21172846
Programme:	MSc in Cloud Computing
Year:	2024
Module:	MSc Cloud Research Project
Supervisor:	Prof. Shivani Jaswal
Submission Due Date:	14/01/2024
Project Title:	A Comparative Analysis of Metaheuristic Algorithms for Optimizing Tasks in Serverless Frameworks for MapReduce Applications
Word Count:	6718
Page Count:	22

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:	Vaishnavi Sarjerao Waghmare
Date:	31st January 2024

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):	

A Comparative Analysis of Metaheuristic Algorithms for Optimizing Tasks in Serverless Frameworks for MapReduce Applications

Vaishnavi Sarjerao Waghmare
21172846

Abstract

In this research, we have proposed a task optimization process for enhancing the performance of MapReduce applications within a serverless framework. The investigation is a comparison of the artificial bee colony algorithm and the Cuckoo Search algorithm in terms of job execution time, CPU resource utilization, and scalability. This analysis utilized the Terasort Hadoop Benchmarking dataset on an AWS EMR instance. The study found that Cuckoo Search Optimization had superior performance compared to ABC in terms of task completion time and CPU Utilization. And when comparing scalability, ABC outperforms Cuckoo Search Optimization Algorithm. These findings demonstrate the effectiveness of the Cuckoo Search Optimization algorithm in reducing task execution times, improving resource utilization, and enhancing adaptability within serverless MapReduce applications. Thus, the paper's novelty illuminates on the potential of nature-inspired algorithms to overcome intrinsic challenges and enhance the efficiency of MapReduce applications. It provides useful insights for the emerging field of Serverless computing.

1 Introduction

Cloud computing is a recent advancement in the IT sector. Cloud computing offers immediate access to extensive pools of scalable computing resources. Due to the increasing number of cloud service providers and improvements in cloud technology, consumers now have access to high-quality services at inexpensive prices. Many firms are transitioning their operations to the cloud due to its adaptable pay-as-you-go pricing model and lack of upfront hardware costs. The regular utilization of such technology results in a substantial accumulation of data. Due to the large amount of data generated by cloud users, a centralized data center consisting of storage devices is necessary. Data management is a crucial operation within the cloud computing environment. Therefore, employing effective job scheduling algorithms has the potential to enhance the velocity and scalability of cloud computing. Thus here we get a narrowed view upon the Serverless Frameworks. MapReduce is a popular programming model used in distributed computing systems such as Hadoop, which divides work into map tasks (to analyze data) and reduce tasks (to consolidate findings). This allows for the processing and generation of massive datasets in parallel. In MapReduce, the term "Serverless" refers to the abstraction of server management, freeing developers from having to worry about managing or providing infrastructure and allowing them to concentrate only on writing code. It is significant

because it simplifies operations, allowing for cost-effective resource allocation and on-demand MapReduce task execution without the need to manage underlying servers. In a serverless MapReduce framework, task optimization is important since it directly affects cost effectiveness and resource consumption. In the dynamically allocated serverless environment, fine-tuning tasks guarantees the best possible allocation of computational demands, lowering execution time and optimizing resource efficiency. In a serverless architecture, this enhancement improves the overall performance, cost-effectiveness, and scalability of MapReduce programs.

The development of distributed computing has led to ground breaking paradigm shift and among them, serverless structures have arisen as a progressive way to deal with application improvement and sending. Serverless computing stands out in this dynamic environment, where agility and scalability are crucial, by removing the complexity of infrastructure management and allowing developers to concentrate solely on writing code, says Balasubramaniam (2017). Therefore this research, focuses on a comprehensive investigation of how MapReduce applications can be integrated into serverless environments, highlighting obstacles and offering suggestions for improving performance. Furthermore, the study compares the use of Artificial Bee Colony and Cuckoo Search optimization algorithm to optimize the sorting process in MapReduce applications.

Problems with Optimizing MapReduce tasks in Serverless despite its promise, integration of serverless environments and MapReduce applications face challenges. The advancement of undertakings experiences difficulties like CPU Resource Usage, cold start issue and Scalability. Exploring these issues requires resourceful solutions for the successful execution of MapReduce applications in serverless systems. The three core problems have been explicitly explained in the further segments of this study. Thus, research tries to establish conjunction identifying the core issues of a serverless architecture and objects to compare the process of task scheduling by comparing the Meta heuristic algorithms utilizing the Hadoop Terasort Benchmark dataset.

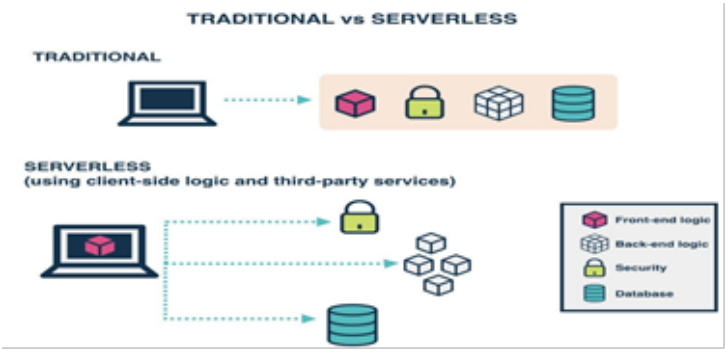


Figure 1: The Serverless Architecture

1.1 Overview of Serverless Frameworks

Serverless architectures, as demonstrated by platforms such as AWS Lambda and Google Cloud functions, fundamentally transform the way applications are designed and run. The essence of serverless computing rests in the event-driven, stateless execution of functions triggered by explicit events. Serverless computing is an attractive option for a wide range of applications due to its planned flexibility, reduced operational overhead, and

cost-effectiveness.
The primary advantage of a serverless architecture lies in the programming flexibility it offers. Unlike traditional or conditional architectures that need programmers to be concerned about backend upkeep, such as handling high workloads, serverless computing may automatically scale in response to incoming requests,says the author Camacho and Alves-Souza (2018). This feature offers a robust scalability capacity to serverless programs, allowing for dynamic resource allocation, hence enhancing flexibility and efficiency.
Furthermore, serverless computing does not imply the absence of servers, but rather the management of servers by service providers like AWS or Google Cloud. This approach is highly efficient for cloud architects, since it enables them to deploy stateless programs and leverage their capabilities. The serverless program has unlimited capabilities, as long as the appropriate resources are carefully selected, allowing the programs to operate until they are entirely executed. Moreover, this capacity to execute actions without storing any information can be implemented in a distributed environment, allowing for parallel processing of the operations. Therefore, a serverless deployment has the ability to adjust to partial failures, allowing the complete application to continue running even if certain components are stopped, is mentioned by the author Nazari et al. (2021).Therefore, by disregarding the program's permanent states, programmers can focus on developing modular and reusable routines to optimize the overall process and improve its efficiency.
However, the extensive adoption of serverless frameworks is mostly driven by their cost-effectiveness, since customers have the option to only pay for the actual computing time utilized by the application during its execution. Clients need not be concerned about the servers' idle time during periods of non-use. Furthermore, there is no need for concern regarding the upkeep of these servers, which makes the serverless design highly cost-effective. This on-demand payment mechanism fulfills the most challenging client needs in their modern business environment, where clients prioritize cost-effectiveness.
The author explains Li et al. (2021) Serverless computing enables the efficient sharing of computing resources among several users in the Cloud by utilizing containers. Every time a function is called, it executes in its own separate container. If a user function does not have an existing container, a new container must be built for it. Nevertheless, the extended duration required for a container to initialize leads to a significant delay in the response time of the operation. Our analysis reveals that the software packages used in the containers for some user operations are largely identical. If an operation necessitating a new container can "borrow" a comparable heated container from earlier actions, it can eliminate the lengthy process of starting from a cold state. Given the aforementioned discovery, we suggest the implementation of Pagurus, a real-time container management system designed to eliminate the issue of cold starting in serverless computing. Pagurus consists of an inter-activity container scheduler and an intra-action container scheduler for each action. The container scheduler facilitates the scheduling of shared containers across different tasks. The intra-action container scheduler is responsible for managing the lifespan of containers. The trial results demonstrate that Pagurus efficiently eradicates the time-consuming process of container cold startup. Pagurus may initiate an action within a few 10 milliseconds, even in the absence of a heated container.

Here, the paper Pu et al. (2019) explains Serverless computing is on the verge of delivering the long-awaited benefits of seamless scalability and price that is calculated in milliseconds. In order to accomplish this objective, service providers enforce a detailed

computational model in which each function is assigned a specific maximum time, a predetermined amount of memory, and no persistent local storage. The precise elasticity of serverless computing is crucial for achieving optimal utilization in general computational tasks, such as analytics workloads. However, the presence of resource limits poses a challenge in implementing applications that require the movement of substantial data between non-overlapping functions. This paper introduces Locus, a serverless analytics system that strategically integrates cost-effective yet slow storage with high-performance yet expensive storage, in order to achieve optimal performance while maintaining cost-efficiency. Locus utilizes a performance model to assist users in determining the appropriate type and quantity of storage needed to achieve the desired balance between cost and performance. We assess the performance of Locus on various analytics applications such as TPC-DS, CloudSort, and Big Data Benchmark. Our evaluation demonstrates that Locus effectively manages the balance between cost and performance, resulting in significant performance enhancements of 4×-500× compared to a baseline system that only uses slow storage. Additionally, Locus reduces resource consumption by up to 59% while achieving similar performance on a cluster of virtual machines. It is worth noting that Locus is 1.99× slower compared to Redshift.

1.2 MapReduce Applications in Serverless Environments

Following its first introduction by Google, the MapReduce programming approach has shown to be highly effective at managing large datasets. These programs are typically used to interface with larger system programs such as Apache Hadoop in order to provide innovative viewpoints. These configurations of serverless programs offer opportunities for efficient and flexible handling and processing of large amounts of data.

MapReduce operates based on the principle of mapping and reducing complex serverless applications. It allows for the mapping and subsequent reduction of multiple processes, enabling the management of intricate tasks through smaller and more manageable subfunctions. The spreading of information among the several distributed hubs combines to produce important results. The mapping and reduction strategy has proven to be highly effective in managing large or intricate datasets where using a single compute machine becomes impractical or restricted is stated by the author Daw et al. (2020). In addition, MapReduce enhances the efficiency of managing and analyzing vast quantities of data by successfully utilizing parallelization to handle complicated processing tasks and optimize computational resources.

Programmers utilize MapReduce in system contexts where dedicated serverless clusters are controlled by frameworks like Apache Hadoop. Nevertheless, the integration of MapReduce functions into a serverless architecture triggers a shift in the prevailing model, creating new opportunities for managing and analyzing large volumes of data. (Kc and Freeh (2014)).

The serverless programs possess a flexible and responsive nature, enabling them to effectively handle data while seamlessly integrating with MapReduce processes. Serverless refers to an AWS Lambda function that enhances programming efficiency in serverless platforms like Amazon and Google. These large-scale suppliers offer extensive capabilities to automate the process of allocating resources and enhance programming through the use of MapReduce. The dynamic scaling feature is highly valuable when workloads fluctuate, which is a common occurrence in big data processing.In addition, the serverless paradigm complements the stateless aspect of MapReduce by allowing capabilities to

be run without restrictions due to explicit events. This aligns with the distributed and parallel characteristics of MapReduce, facilitating the smooth integration of these two potent paradigms. (Lloyd et al. (2018))

1.3 Challenges in Optimizing Tasks in Serverless for MapReduce

Calibrating resource allocation to match the specific requirements of MapReduce jobs in serverless platforms can be a difficult task. The diverse computing and storage demands of different stages in a MapReduce operation make it challenging to effectively allocate work among available resources.

The inherent decentralization, adaptability, and self-organization of these algorithms make them particularly suitable for addressing the complex challenges of resource allocation, the problem of starting from scratch, and the fluctuation of dynamic workloads in relation to MapReduce functions within a serverless architecture.

These methods will be used to repeatedly improve the allocation of assets, aiming to achieve an optimal distribution of tasks across the serverless environment. This method possesses the capacity to flexibly adjust to different workloads and enhance the distribution of computational resources by mimicking the collaborative and decentralized decision-making process observed in bee colonies, specifically tailored to the specific requirements of MapReduce applications.

1.4 Research Objective

Thus, considering the limitaions of the default tasks optimizations or the traditional ways of optimizations for MapReduce particularly in relation with the meta-heuristic algorithms of the Serverless Frameworks , this research delves into the tasks optimization by considering few key metrics like Execution time, CPU resource utilization and Scalability. While we know there are various approaches to develop a custom algorithm, the meta-heuristic approaches have emerged as the most commonly used algorithm in this context, which brings us to our research question-
I would also like to specify about the novelty of the proposed research. As highlighted in the related works many researchers have done task optimization using various platforms like Cloud, Fog and Edge or let that be in Serverless, but here comes a point where I understood the research gap in numerous papers which said, further the research could be carried out to find the latencies, scalability or the resource usage time more precisely using various optimization algorithms. Hence, which gave me a ray and directed to conduct this research in Serverless platform and using the ABC algorithms as none study was yet found to be done specifically in Serverless using these two algorithm. Hence, we come to our research question:

How does the performance and efficiency of optimizing tasks in Serverless Frameworks for MapReduce applications improve when applying the meta-heuristic algorithms, and how well does these algorithms compare in terms of execution time, CPU utilization and scalability?

1.5 Structure and Outline

The remaining sections of this report are divided in few sections as mentioned below:

- Section 2 represents the related works that have been implemented related to the topic Optimizing Tasks in Serverless and MapReduce in Serverless Frameworks.
- Section 3 of the paper gives us the details about the methodology followed.
- Section 4 gives a quick overview of the design and implementation of this work.
- Section 5 presents the experiments carried out and its results.
- Section 6 is the last section which provides us with the conclusion and future work.

2 Related Work

The presentation by Gu et al. (2014) provides an overview of the latest developments in serverless computing. As per the author's statement, serverless technology has eradicated the necessity for developers to allocate and oversee servers. In order to attain the necessary business logic, developers should prioritize functions over business logic. The author delineates the disadvantages of utilizing serverless architecture for both consumers and suppliers. FaaS models impose some constraints on consumers, such as the absence of the most up-to-date versions of software libraries. Delivering services entails overseeing the whole lifespan of a user's operation, including its ability to handle increased demands and its resilience to failures. Developers will have the ability to create applications that are specifically tailored to the behavior of the platform. The author highlights many issues for both providers and users, including cost, cold start, resource restrictions, security, scaling, hybrid cloud, and legacy system challenges. In addition to lowering the cost of resource utilization during job execution and idle periods, Serverless encounters a basic difficulty related to pricing. In order to minimize expenses, it is necessary to implement an efficient scaling mechanism that would completely eliminate the charge during periods of inactivity. In order to mitigate resource constraints, it is imperative to effectively administer the platform during periods of increased demand and in the face of potential cyber threats. Given that multi-user platforms perform numerous activities, it is imperative for the provider to effectively adjust and segregate these functions. The supplier must ensure that functions are both elastic and scalable as appropriate. Developers face several issues while working with serverless architecture, including configuration, deployment, IDEs, concurrency, scalability, monitoring, and debugging.
The study asserts that a trade-off between memory and latency is essential in serverless settings. The EMARS approach seeks to optimize this trade-off by allocating memory according to the individual requirements of functions, hence improving efficiency in serverless platforms.

The study by Saha and Jindal (2018)signifies a notable advancement in the effective management of resources in serverless computing, particularly focusing on the difficulties related to memory allocation. Moreover, it presents numerous opportunities for future research, specifically in the integration of these solutions with established serverless platforms and the optimization of these solutions according to practical workloads. This study

presents a mechanism that enables the monitoring of memory and workload in order to accurately estimate future requirements. This code dynamically sets memory restrictions in our present implementation, which operates independently of OpenLambda. To achieve an urgent objective, it is necessary to connect this dynamic update to memory with OpenLambda in order to provide a comprehensive solution. The config generator can be enhanced with intelligence to determine the most efficient memory limitations by analyzing the logged information. In order to perform workload based modeling, it is necessary to determine a threshold value based on additional testing. Furthermore, we intend to enhance the assessment process by conducting tests using authentic workloads and evaluating the performance on alternative serverless platforms that are open-source and using different heuristic and metaheuristic algorithms, Another potential avenue for future research is the utilization of machine learning methodologies to forecast the most suitable memory needs.

The author Silambarasan et al. (2016) says, ensuring optimal usage and performance isolation of basic hardware while managing resources at a large scale is a crucial challenge for any cloud management software. In addition to the scalability problem, a cloud-level resource management layer needs to address the heterogeneity of frameworks, compatibility requirements between virtual machines and underlying hardware, the creation of isolated resource islands due to storage and network connectivity, and the limited capacity of storage resources. In this study, we conducted an investigation in multiple fields to explore potential opportunities for optimization. The main objective was to develop an efficient topology to address resource allocation issues in the resource model. The Artificial Bee Colony Algorithm (ABC) was chosen as the optimization algorithm for the multi-objective problem. It has been found to provide the best optimized result and requires less computation time to achieve the desired objective. The results of the proposed topology have shown promising and effective outcomes, with reduced computational effort. This paper further says to increase the efficiency of the experiments conducted in the Serverless platform.

In this paper the author Liu et al. (2014) focuses on the issue of efficient task scheduling in cloud computing, which is crucial for maximizing resource usage and minimizing execution time. The proposal introduces a new scheduling technique that synergistically combines the advantages of Genetic Algorithms (GA) with Ant Colony Optimization (ACO). This strategy utilizes the global search capability of GA to rapidly identify optimal solutions and the efficient convergence of ACO through strong positive feedback. The approach commences by employing a Genetic approach (GA) to produce a collection of optimal solutions. These solutions are subsequently utilized as the initial pheromones for the Ant Colony Optimization (ACO) process. The objective of this hybrid approach is to address the drawbacks of both GA (inefficiency in late-stage optimization) and ACO (dependence on starting pheromone levels). The algorithm's usefulness is demonstrated through simulation studies, particularly in large-scale systems, as compared to employing GA or ACO alone. The results indicate that the GA-ACO algorithm surpasses both GA and ACO in terms of success rate and the amount of iterations needed to discover the best solution. The research emphasizes the algorithm's superior efficiency in cloud computing environments, where the management of task scheduling and resource allocation is intricate. The combination of Genetic Algorithm (GA) and Ant Colony Optimization (ACO) shows advantageous for scheduling tasks in cloud computing, enhancing the algorithm's search efficiency. The research indicates the possibility of further investigation

and implementation of combination optimization strategies in cloud computing or serverless systems, utilizing diverse optimizing algorithms.

In this paper the author Ustiugov et al. (2021) evaluated the latency of the cold start for each capability and compared it to the latency of the warm capability. It is important to note that when a warm instance is called, it stays in memory and does not encounter any delay when starting up. In order to obtain a detailed breakdown of the latency experienced during the initial start of the system, their instrument invoked each capacity many times. To simulate a cold latency, they clear the page caches of the host operating system after each test. Figure 2 illustrates the latency for both the cold and warm serving of each function. A function instance exhibits little request latency when it remains in a warm state, meaning it is stored in memory, as anticipated. In contrast, the authors discovered that initiating a process from a starting point that lacks prior information takes one to two considerably longer periods of time compared to initiating it from a point where prior information is available. This finding illustrates that despite the use of advanced snapshotting techniques, delays in cold starts remain a significant issue for these functions.

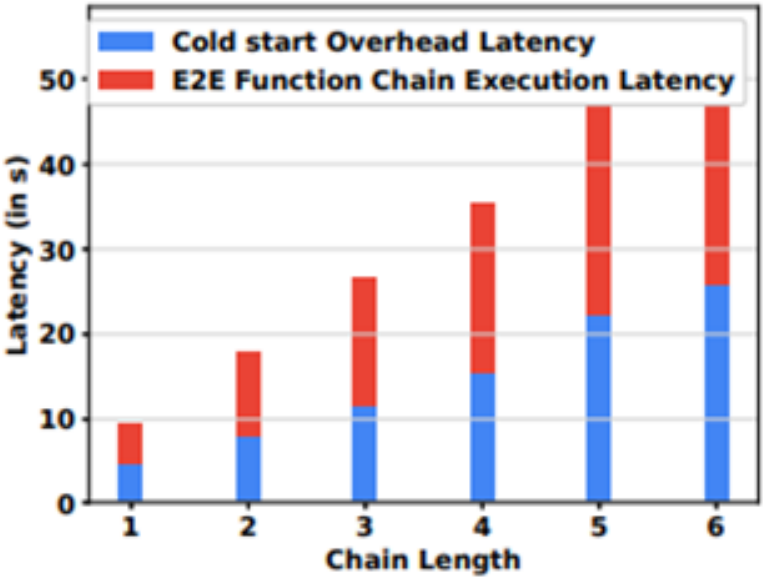


Figure 2: Cold Start latency versus chain length of the function Ustiugov et al. (2021)

In the research the author Lee et al. (2018) explains, Serverless computing offers a compact runtime environment for executing code without the need for infrastructure administration. It is similar to Platform as a Service (PaaS), but operates at a more granular level. Amazon introduced Lambda functions, an event-driven compute service, in 2014 with a restriction of 25 concurrent invocations. However, it now allows for a minimum of one thousand concurrent invocations to handle event messages created by various resources such as databases, storage, and system logs. Google, Microsoft, and IBM, among other providers, offer a dynamic scaling manager to efficiently handle simultaneous requests for stateless operations. This involves installing extra containers on new computing nodes for distribution. Functions are commonly designed for microservices and

lightweight workloads, and they are closely linked to distributed data processing through concurrent invocations. This paper explains that the existing serverless computing environments are capable of concurrently executing dynamic applications, provided that a partitioned job may be executed on a tiny function instance. This paper provides the findings of the study on the performance of concurrent invocations, specifically in terms of throughput, network bandwidth, file I/O, and computation performance and implemented a set of distributed data processing functions to handle scalability, and subsequently compared the cost effectiveness and resource consumption of serverless computing and virtual machines. Serverless computing presently employs containers with limited computing resources for temporary workloads. However, the paper anticipates that in the future, there will be a wider range of computing resources available with fewer limitations on configurations. This will enable the handling of intricate workloads by optimizing and applying various algorithms.

In this paper, the author Giménez-Alventosa et al. (2019) explains, MapReduce is a highly popular programming paradigm utilized for analyzing vast datasets, sometimes referred to as Big Data. Serverless computing, specifically Functions as a Service (FaaS), has become increasingly popular as an execution model. It eliminates the need for users to directly manage servers, such as virtual machines. Contrarily, the Cloud provider assigns resources to function invocations in a flexible manner and implements detailed pricing based on the duration of execution and allocated memory, as demonstrated by AWS Lambda. This article presents the development of a very efficient serverless architecture for running MapReduce tasks on AWS Lambda, with Amazon S3 serving as the storage backend. Furthermore, a comprehensive evaluation has been conducted to examine the appropriateness of AWS Lambda as a platform for executing High Throughput Computing tasks. The findings suggest that AWS Lambda offers a user-friendly computing platform for versatile applications that meet the service's limitations (maximum execution time of 15 minutes, 3008 MB of RAM, and 512 MB of disk space). However, it displays inconsistent performance patterns that could hinder its acceptance for closely interconnected computing tasks. In future the challenges in this paper could be took into broad spectrum considerations by incorporating various optimization algorithms.

In this study by Kazimov (n.d.), Cloud computing is a framework that allows for widespread, convenient, and instant access to a shared collection of customizable computing resources (such as networks, servers, storage, applications, and services). These resources can be quickly allocated and released with minimal effort or involvement from the service provider. Functions as a Service, also known as Serverless computing, is a recently announced idea by Amazon in 2015. It is an execution model in which the cloud provider is responsible for running a specific piece of code by dynamically allocating the necessary resources. This study presents a comprehensive analysis of the constraints associated with the serverless cloud computing concept. Initially, the problem encountered during the initial execution of the task required more time compared to the typical execution time. This situation led to the realization that instead of delivering the data individually, it would be more efficient to combine the data beforehand.

Here, the author Suresh et al. (2020) says there is a pressing issue in the realm of serverless computing is the increasing expense of the infrastructure required to manage the expanding volume of traffic at a large scale. This paper introduces ENSURE, a sched-

uler and resource manager that operates at the function level. Its purpose is to minimize the price of resources for providers while still achieving the performance requirements of customers. ENSURE operates by categorizing incoming function requests during execution and effectively managing the allocation of resources for colocated functions on each invoker. ENSURE is designed to dynamically adjust its capacity based on workload traffic, utilizing principles from operations research, in order to eliminate cold starts. This allows for efficient scaling to meet changing demands. Ultimately, the goal is to optimize the scheduling of requests by focusing the workload on a sufficient number of invokers. This approach promotes the reuse of active hosts, hence minimizing the occurrence of cold starts. Additionally, it enables unnecessary capacity to be efficiently and smoothly terminated when it is no longer needed. We apply the ENSURE technique to Apache OpenWhisk and demonstrate that it substantially enhances resource efficiency, up to 52%, when compared to previous benchmarks. Moreover, it maintains acceptable application latency. The author of this study also discusses the potential future deployment of utilizing diverse optimization strategies across multiple platforms.

3 Methodology

This section outlines the assessment plan for exploring the optimization of MapReduce capabilities in serverless systems. The thought process incorporates the certainty of serverless architectures, the deployment of MapReduce applications, and the overall approach to handling experimentation and analysis.

3.1 MapReduce Application Design

The incorporation of MapReduce applications for trial and error is an essential component of the exploration strategy. This involves creating delegate MapReduce jobs that exemplify changing computational and storage requirements. The applications will be designed to replicate real-life scenarios, including issues such as data volume, algorithm complexity, and dynamic workload variations. Special attention will be devoted to addressing challenges associated with statelessness, cold start latency, and dynamic workload fluctuations within the context of serverless architectures.

The method focused on optimizing the algorithms ABC and Cuckoo Search by measuring their performance on EMR instances as core Hadoop jobs. The optimization was varied, and the optimized result was multiplied by 3X to further enhance the optimization. The study aims to measure job completion durations, compare the scalability of algorithms, and analyze resource utilization. To achieve this, the study suggests utilizing the widely used "Rosenbrock function" for work scheduling and benchmarking purposes.

3.2 Data-set

3.2.1 Benchmarking Hadoop MapReduce using TeraSort

Hadoop TeraSort is a renowned benchmark dataset designed to efficiently sort 1 TB of data using Hadoop MapReduce. The TeraSort benchmark rigorously tests all components of the Hadoop MapReduce framework and the HDFS filesystem, making it an excellent option for optimizing the design of a Hadoop cluster.

The initial TeraSort benchmark dataset arranges 10 million records, each consisting of 100 bytes, resulting in a total data size of 1 terabyte (TB). Nevertheless, we have the capability to designate the quantity of records, enabling us to customize the overall magnitude of data. [1]

4 Design Specification

This section will cover the Meta-heuristic algorithms used, their general working and why we have chosen these algorithms for our implementation. Also we shall discuss the architecture diagram proposed for this research and the Rosenbrock function used for scalability.

4.1 Architecture Diagram

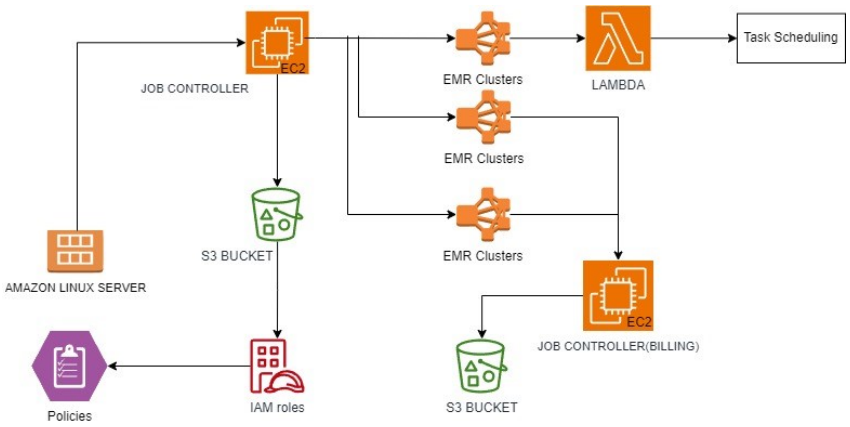


Figure 3: System architecture for MapReduce

The architecture diagram, Figure 5 illustrates a sophisticated Amazon Web Services (AWS) system designed for work control and scheduling. The system utilizes many AWS services like as EC2, EMR instances, Lambda, S3 bucketss, IAM jobs, and methodologies. The Job Controller, situated on an EC2 instance, serves as the principal component. EC2 (Elastic Compute Cloud) provides the ability to adjust the capacity of computing resources in the cloud, allowing users to operate virtual servers.

[1]: Benchmarking Hadoop MapReduce using TeraSort`https://subscription.packtpub.com/book/data/9781783285471/1/ch01lvl1sec18/benchmarking-hadoop-mapreduce-using-terasort`

The Job Controller is linked to an Amazon Linux Server and an S3 buckets. The Amazon Linux Server is a proven enterprise-grade platform that offers robust security and extensive support for a wide range of applications. The S3 bucket is employed for storing and retrieving any quantity of data at any time; it functions as a global file system. IAM jobs are closely linked to the Job Controller. IAM enables secure management of access to AWS services and resources for users. The IAM jobs are associated with approaches that define permissions to determine the actions that can be executed on specific resources.

On the right side of the chart, Lambda is employed for task scheduling. AWS Lambda enables serverless execution of code, eliminating the need for server installation and management. It automatically scales applications by executing code in response to each trigger. The EMR clusters are displayed as being linked to both the Job Controller and Lambda. EMR is a managed cluster platform designed to facilitate the execution of big data frameworks such as Apache Hadoop and Apache Spark on Amazon Web Services (AWS). It enables the processing of large volumes of data by utilizing resizable clusters of Amazon EC2 instances.

We have created two Lambda functions for task scheduling. The jobs are scheduled using two algorithms inside these two lambda functions. The dataset is stored in one of the S3 bucket and the other stores jobs logs. The bottom right corner of the design displays the segregation of duties or specific responsibilities, allowing for the autonomous handling of charging-related roles from other types of positions to ensure efficiency and security.

This architecture utilizes various AWS services. EC2 instances serve as Job Controllers to ensure efficient administration. S3 buckets provide flexible storage solutions. IAM roles ensure secure access management governed by combined policies. Lambda enables automated task scheduling without manual intervention. EMR clusters offer powerful platforms for processing large-scale data. Each component plays a crucial role in ensuring efficient project planning, effective job control, and adherence to optimal security protocols through IAM roles and policies, ensuring seamless yet secure operations inside this cloud-based environment.

4.2 Meta-heuristic Algorithms

Metaheuristic algorithms play a major role in optimization by effectively exploring solution spaces to discover solutions that are almost optimal. Applications of these techniques extend beyond classic NP-complete problems, encompassing areas such as optimization, scheduling, and routing. These algorithms imitate processes such as evolution, swarm behavior, and other intelligent systems. Some examples of optimization techniques are colony optimization, simulated annealing, genetic algorithms, and particle swarm optimization. Metaheuristic algorithms are adept at addressing problems by successfully managing the trade-off between exploration and exploitation. This makes them especially valuable in situations that are ambiguous or computationally demanding. Metaheuristic algorithms are beneficial for large-scale operations as they support multi-objective optimization and produce optimal, if not accurate, solutions. Utilizing metaheuristic techniques can effectively address the distinct objectives of minimizing latency and maximizing utilization. This study aims to compare two metaheuristic algorithms.

4.3 Artificial Bee Colony

The Artificial Bee Colony algorithm is an optimization technique inspired by the foraging behavior of bumble bee colonies. Introduced by Karaboga (2010), ABC is significant for the broader category of population-based algorithms, deriving inspiration from the collective behavior observed in social insect states, particularly bumble bee colonies. The ABC algorithm represents the iterative process of a honey bee colony searching for the optimal food source to produce honey, which is then applied to a specific problem. The algorithm consists of three main components: scout bees, employed bees, and observer bees. Each of these honey bee species plays a distinct role in the pursuit of optimal solutions.

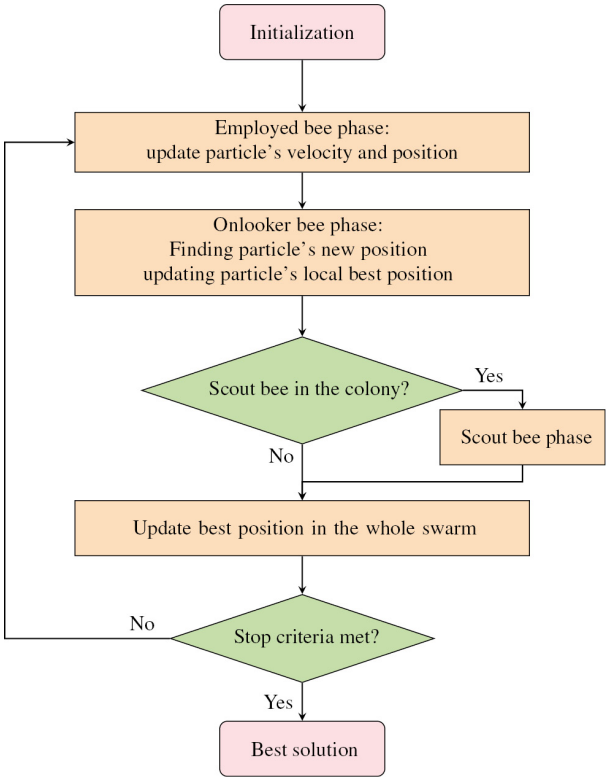


Figure 4: Flowchart of ABC algorithm working

4.4 Cuckoo Search Algorithm

The Cuckoo Search technique (CSA) is an optimization technique inspired by the reproductive behavior of cuckoo birds. The technique, introduced by Yang and Deb (2014), aims to address optimization problems by simulating the obligate brood parasitism behavior observed in certain cuckoo species. These avian species deposit their eggs in the nests of other bird species, relying on the hosts to incubate and rear their offspring.The fundamental concept underlying the Cuckoo Search method is to treat potential solutions to an optimization problem as eggs in a nest. Each egg corresponds to a solution, and the quality of a solution is evaluated based on the optimization function. During the optimization process, nests with enhanced configurations are considered more attractive, analogous to how a cuckoo would prefer a nest with eggs that had a higher likelihood of

survival. The Cuckoo Search algorithm has been utilized in a wide range of optimization applications, such as engineering design, parameter tweaking, and machine learning. It is renowned for its straightforwardness, straightforwardness of execution, and optimal balance between exploration and exploitation. Nevertheless, the performance of the system can fluctuate based on the specific attributes of the optimization problem.

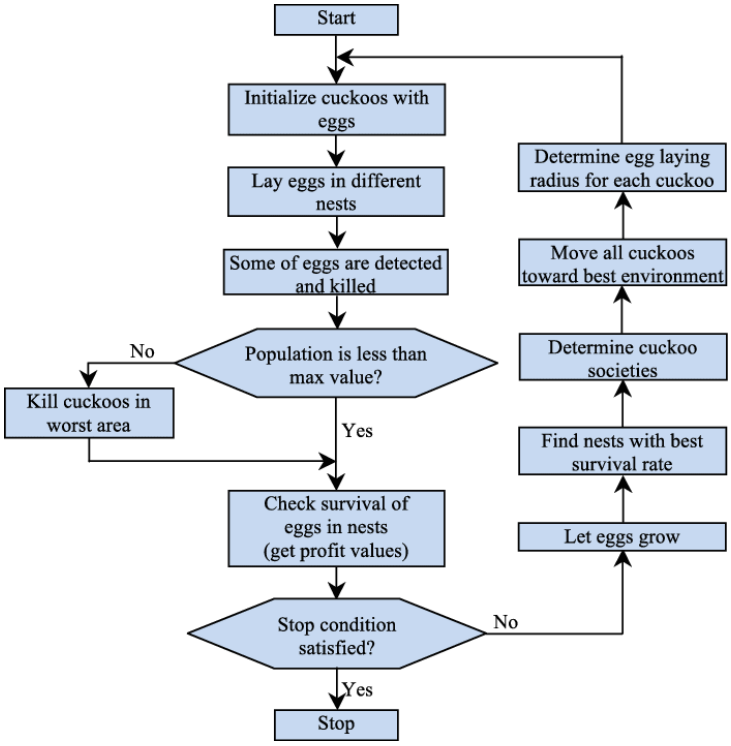


Figure 5: Flowchart of Cuckoo Search algorithm working

4.5 Discussion on Algorithms:

The ABC algorithm has demonstrated effective use in several optimization issues, such as function optimization, machine learning, and software design. The popularity of computational intelligence is attributed to its simplicity, flexibility, and ability to handle both continuous and discrete optimization problems.

The Cuckoo Search algorithm is renowned for its efficacy in identifying superior solutions to diverse optimization problems due to its straightforwardness and user-friendliness. The intriguing and innate behavior of cuckoo birds serves as a compelling and organic approach to innovation, offering an alternative tool for experts and practitioners in industries such as engineering, finance, and machine learning.

4.6 Rosenbrock Function

The Rosenbrock functions, also called the Rosenbrock's valley or banana functions are a normally involved numerical functions for testing optimization problems. It is many

times utilized as a benchmark function because of its characteristic valley curve. The Rosenbrock function for two factors (n=2) is

$f(x, y) = (a - x)^2 + b * (y - x2)^2$

It has global minimum at (a, a^2) where, a is regularly set to 1, and b is set to 100. The capability represents a test to improvement calculations in light of the fact that the global minimum is inside a long, narrow, parabolic formed flat valley curve. Efficient optimization requires crossing this valley transversely. These functions are used as benchmark for calculating and evaluating the performance of two algorithms namely Artificial Bee Colony and Cuckoo Search.

5 Implementation

In this section we shall see the implementation of ABC and Cuckoo Search and the lambda functions created for both the algorithms.

5.1 Artificial Bee Colony Optimization Algorithm Implementation

A crucial aspect of this study involves implementing the Artificial Bee Colony (ABC) Algorithm to enhance the efficiency of MapReduce workloads within serverless frameworks. The subsequent sections delineate the fundamental processes and concerns of the ABC Algorithm specifically tailored for the serverless context. The ABC algorithm is designed to accommodate the unique characteristics of serverless deployment by anticipating variations. To do this, it is necessary to synchronize the various elements of the algorithm with stateless serverless functions in an event-driven manner. The ABC Algorithm will be adapted to address resource allocation and cold start challenges specific to MapReduce operations in serverless infrastructures.

The enhanced ABC algorithm will be seamlessly integrated into selected serverless systems, ensuring compatibility with their respective APIs and event triggers. Each serverless capacity will operate as a prospective candidate within the ABC algorithm, with the algorithm coordinating the analysis and detection of these candidates to increase the MapReduce function. The combination will involve leveraging serverless platform features for automated scalability and efficient resource management.

An essential aspect of the ABC algorithm implementation is the distinct allocation of resources based on various MapReduce tasks. The method will efficiently distribute jobs among the serverless functions, considering their current workload and computing capacity. The objective of this dynamic resource allocation is to effectively adjust to the evolving requirements of MapReduce applications while optimizing the execution efficiency of functions.

Solving the cold start problem is crucial for the performance of the ABC algorithm in serverless environments. Techniques such as capability pre-warming or intelligent deployment features will be implemented to reduce the latency associated with cold starts. The ABC Algorithm will integrate these strategies into its decision-making process to ensure timely and efficient execution of MapReduce workloads.

The steps for the implementation are as follows-

```
perform_abc_optimization() {
    # Parameters
    max_iterations=100
    population_size=20
    employed_bees=$((population_size / 2))
    onlooker_bees=$((population_size / 2))

    # Initialization
    best_solution=$((RANDOM % 10000 + 5000))
    # Generate random initial solutions
    solutions=()
    for ((i = 0; i < population_size; i++)); do
        solution=$((RANDOM % 10000 + 5000))
        solutions+=($solution)
    done

    # ABC optimization loop
    for ((iteration = 0; iteration < max_iterations; iteration++)); do
        # Employed bees phase
```

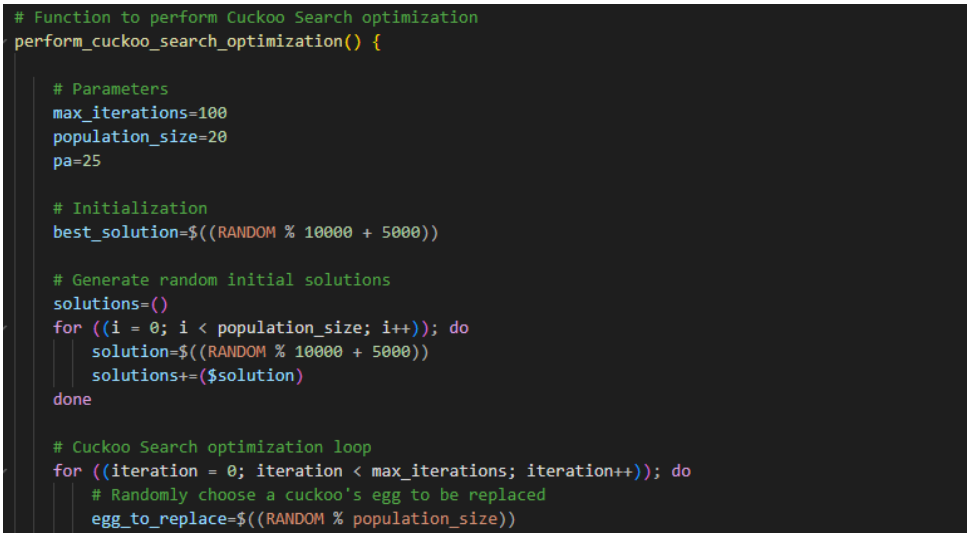
Figure 6: ABC Optimization function

Firstly, we choose and create an EC2 instance by choosing the kernel and instance type, next followed by creating a secure key pair and then launching the instance and accessing the SSH via the powershell, then configuring the EMR instance through putty and benchmarking and lastly configure the Figure 6 ABC lambda function for comparison and the further steps for Cuckoo Search are mentioned in the next section.

5.2 Cuckoo Search Optimization Algorithm Implementation

The implementation of the Cuckoo Bird Optimization technique to enhance MapReduce capabilities within serverless designs is a crucial and precise process. This section provides a comprehensive depiction of the coordination and application of the Cuckoo Bird Algorithm for task optimization. The primary phase involves integrating the Cuckoo Bird optimization algorithm into selected serverless frameworks such as AWS Lambda. This integration necessitates modifying the algorithm to align with the event-driven and stateless characteristics of serverless computing. The integration of the Cuckoo Bird Algorithm into the MapReduce application architecture will enable it to efficiently allocate resources and tasks based on real-time events. The execution of the Cuckoo Bird algorithm focuses on efficient resource allocation, which is a vital aspect of adapting to the changing demands of MapReduce capabilities. The algorithm consistently assesses the storage and computational capacities and progressively adjusts resource allocation to enhance execution. The serverless environment can effectively allocate resources based on dynamic computational demands, thanks to its flexibility.

Now, we choose and create an EC2 instance by choosing the kernel and instance type, next followed by creating a secure key pair and then launching the instance and accessing the SSH via the powershell, then configuring the EMR instance through putty and benchmarking after configuring the ABC Lambda function for comparison, likewise we have to configure the Cuckoo Search Lambda function as shown in Figure 7 for comparison. All the required screenshots are presented well in the configuration manual.



```
# Function to perform Cuckoo Search optimization
perform_cuckoo_search_optimization() {

    # Parameters
    max_iterations=100
    population_size=20
    pa=25

    # Initialization
    best_solution=$((RANDOM % 10000 + 5000))

    # Generate random initial solutions
    solutions=()
    for ((i = 0; i < population_size; i++)); do
        solution=$((RANDOM % 10000 + 5000))
        solutions+=($solution)
    done

    # Cuckoo Search optimization loop
    for ((iteration = 0; iteration < max_iterations; iteration++)); do
        # Randomly choose a cuckoo's egg to be replaced
        egg_to_replace=$((RANDOM % population_size))
```

Figure 7: Cuckoo Search Optimization function

6 Evaluation

The evaluation of results conducted will be explained in this section. The experiment has been conducted initially and finally on both the algorithms, which means the initial one is done with the default Rosenbrock function and the results got are then multiplied 3X times to find scalability. Along with that a comparative analysis of both the algorithms with reference to Execution time, CPU resource utilization and Scalability have been conducted.

6.1 Execution Time

Figure 8 depicts the execution time comparison of two algorithms namely Artificial Bee Colony(ABC) and Cuckoo Search Optimization. The x-axis denotes different algorithms and y-axis denotes time taken in seconds. These algorithms are compared on the basis of six parameters.

- Cold Start: It refers to time taken by a serverless function to initialize and start executing in response to an event.
- Optimization Time: The time and effort spent in optimizing the performance of the MapReduce job.
- TeraGen Time: It refers to a benchmark in Hadoop system to generate a large amount of synthetic data.
- TeraSort Time: Time taken to sort generated data using MapReduce function.
- TeraValidate Time: Benchmark designed to validate the sorted order of data.
- Total time: The total execution time taken.

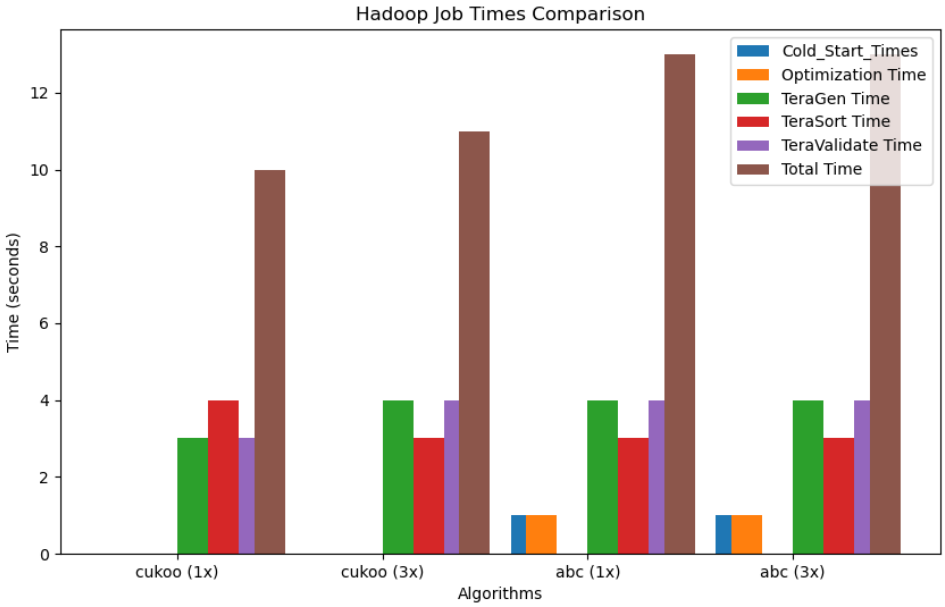


Figure 8: Execution time of ABC vs. Cuckoo Search Optimization Algorithm

Note: In all the above different parameters low is better.

From Figure 8, it can be seen that Cuckoo Search algorithm has no cold start and optimization time as compared to ABC algorithm. TeraGen time for cuckoo algorithm increases from 1x to 3x(3s to 4s approx) and for ABC algorithm it is constant at 4s. TeraSort time decreases from 4s to 3s approx for Cuckoo algorithm when testing from 1x to 3x. And for ABC algorithm it is constant 3seconds. TeraValidate time of cuckoo algorithm is 3 seconds which increases to 4 seconds when increasing from 1x to 3x. For ABC algorithm, it is constant 4 seconds. Total execution time of Cuckoo search is less than ABC algorithm by 3 seconds approximately.

6.2 CPU Resource Utilization

Figures 9 and 10 depicts 4 different graphs where x-axis denote Measurements and y-axis denotes the percentage. The algorithms are compared on the basis of four different parameteres.

- CPU Usage: Total percentage of the CPU utilized.
- System Space Utilization: Total percentage of system space utilized.
- Idle time: Percentage of CPU time in idle state.
- Borrowed time: CPU time taken for a VM.

Figure 9 shows the CPU Resource Utilization for ABC algorithm and figure 10 Shows CPU Resource Utilization for Cuckoo Algorithm. Comparing all the four parameters of these two algorithms, it can be seen that out of all the parameters, Borrowed time of these two algorithms differ with Cuckoo Algorithm having less percentage of borrowed time than ABC algorithm.

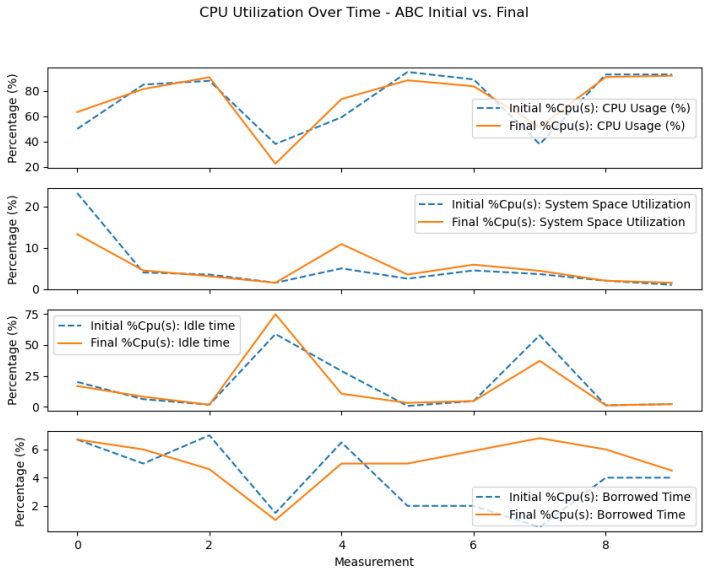


Figure 9: CPU Resource Utilization for ABC Algorithm before and after fine tuning optimization

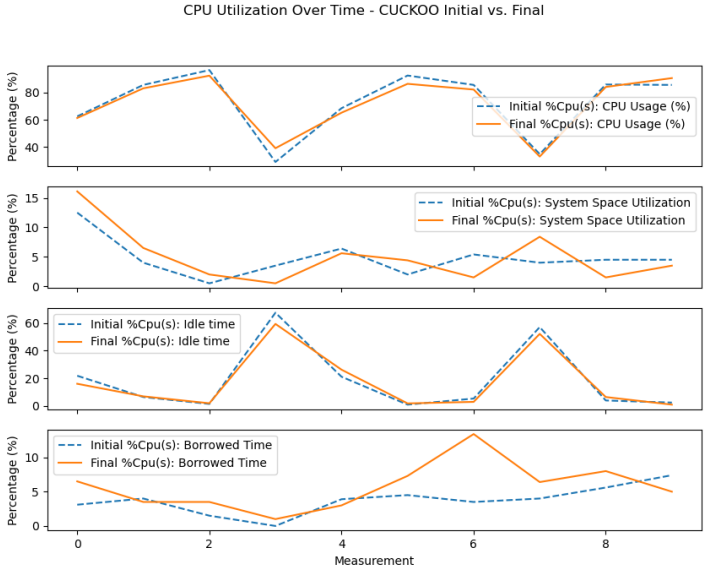


Figure 10: CPU Resource Utilization for Cuckoo Algorithm before and after fine tuning optimization

6.3 Scalability

In the Figure 11 demonstrates the scalability comparison of the two algorithms in which the ABC algorithm performed better than the cuckoo algorithm as was observed to lower the resource consumption upon fine tuning, while the cuckoo algorithm enhanced the resource usage.

Figure 11 denotes Scalability comparison between ABC and Cuckoo Search Algorithms on the basis of four parameters.

- User Time(us): Percentage of CPU Utilization time on executing user processes.
- System Time(sy): Percentage of CPU time on executing system level(kernel)
- Idle Time(id): CPU percentage in idle state.
- Steal Time(st): Percentage of time taken from virtual machine by hypervisor.

From Figure 11, it can be clearly seen that in both test (Initially and Finally), ABC algorithm performs better than Cuckoo Algorithm. From the graph of Initial Scalability Analysis, it can be seen that ABC algorithm is 10 percent efficient than Cuckoo Search with respect to User time and Cuckoo Search Algorithm has more idle time than ABC algorithm. And approximate same difference of 10 percent approximate can be observed with respect to idle time.

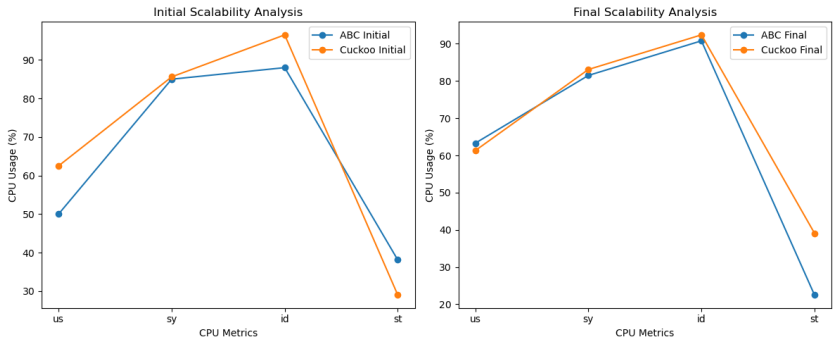


Figure 11: Scalability comparison of ABC and Cuckoo Search algorithms

6.4 Discussion

In this section, lets discuss the evaluation of these experiments

- Figure 8 shows that the Cuckoo Search Algorithm takes approximately 23% less total time to execute than ABC algorithm. In the case of Cuckoo(1x) is the lowest as compared to ABC(1x) and ABC(3x) by 10 percent approximately which indicates that Cuckoo Search Algorithm performs better than ABC Algorithm with respect to execution time.
- From Figures 9 & 10, it can be seen that both algorithms perform in a similar manner with an exception with respect to borrowed time. When comparing borrowed time, Cuckoo Search Algorithm has lowest borrow time than ABC algorithm which is preferred. Thus in CPU utilization Cuckoo Algorithm performs the same as ABC Algorithm but is slightly better in certain conditions.

- From Figure 11, it can be seen that during initial Scalability test ABC algorithm has the least user processes execution time(us) than Cuckoo Algorithm with a 10 percent less utilization. And in the Final Scalability graph, both algorithm performs in a similar manner with ABC algorithm outperforming Cuckoo Search Algorithm slightly. Thus ABC algorithm is preferred over Cuckoo Search Algorithm with respect to Scalability.

7 Conclusion and Future Work

In this paper, a comparative analysis between two meta-heuristic algorithms namely Artificial Bee Colony Algorithm and Cuckoo Search Optimization Algorithm with respect to MapReduce Applications in serverless platform has been performed successfully. The recommended Cuckoo Search Optimization Algorithm performs better than Artificial Bee Colony Algorithm with respect to execution time but when scaling up it can be observed that ABC algorithm performs better than Cuckoo Search algorithm. And when comparing CPU utilization with respect to MapReduce applications, both algorithms perform in a similar manner with Cuckoo Search being slightly more optimized than ABC algorithm. These experiments were conducted on AWS Lambda i.e., a serverless platform, the future works inlude:

- Refining the current algorithms and investigating hybrid methodologies that consolidate various streamlining procedures.
- Considering other factors to improve task enhancement in Serverless MapReduce applications.
- Analyzing and comparing different algorithms other than meta-heuristic to optimize task in MapReduce applications.

References

Balasubramaniam, K. V. (2017). *System and Analysis for Low Latency Video Processing using Microservices*, University of California, San Diego.

Camacho, L. A. G. and Alves-Souza, S. N. (2018). Social network data to alleviate cold-start in recommender system: A systematic review, *Information Processing & Management* **54**(4): 529–544.

Daw, N., Bellur, U. and Kulkarni, P. (2020). Xanadu: Mitigating cascading cold starts in serverless function chain deployments, *Proceedings of the 21st International Middleware Conference*, pp. 356–370.

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.

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.

Karaboga, D. (2010). Artificial bee colony algorithm, *scholarpedia* **5**(3): 6915.

Kazimov, S. (n.d.). Limitations of serverless computing.

Kc, K. and Freeh, V. W. (2014). Dynamic performance tuning of hadoop, *Technical Report, North Carolina State University* .

Lee, H., Satyam, K. and Fox, G. (2018). Evaluation of production serverless computing environments, pp. 442–450.

Li, Z., Chen, Q. and Guo, M. (2021). Pagurus: Eliminating cold startup in serverless computing with inter-action container sharing, *arXiv preprint arXiv:2108.11240* .

Liu, C.-Y., Zou, C.-M. 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*, IEEE, pp. 68–72.

Lloyd, W., Vu, M., Zhang, B., David, O. and Leavesley, G. (2018). Improving application migration to serverless computing platforms: Latency mitigation with keep-alive workloads, *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, IEEE, pp. 195–200.

Nazari, M., Goodarzy, S., Keller, E., Rozner, E. and Mishra, S. (2021). Optimizing and extending serverless platforms: A survey, *2021 Eighth International Conference on Software Defined Systems (SDS)*, IEEE, pp. 1–8.

Pu, Q., Venkataraman, S. and Stoica, I. (2019). Shuffling, fast and slow: Scalable analytics on serverless infrastructure, *16th USENIX symposium on networked systems design and implementation (NSDI 19)*, pp. 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)*, IEEE, pp. 827–830.

Silambarasan, K., Ambareesh, S. and Koteeswaran, S. (2016). Artificial bee colony with map reducing technique for solving resource problems in clouds, *Indian Journal of Science and Technology* **9**(3): 1–6.

Suresh, A., Somashekar, G., Varadarajan, A., Kakarla, V. R., Upadhyay, H. and Gandhi, A. (2020). Ensure: Efficient scheduling and autonomous resource management in serverless environments, *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, IEEE, pp. 1–10.

Ustiugov, D., Petrov, P., Kogias, M., Bugnion, E. and Grot, B. (2021). Benchmarking, analysis, and optimization of serverless function snapshots, *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 559–572.

Yang, X.-S. and Deb, S. (2014). Cuckoo search: recent advances and applications, *Neural Computing and applications* **24**: 169–174.