Developing ReAdaBalancer for Load Balancing Optimization in Networked Cloud Computing

M Diarmansyah Batubara[1]*, Poltak Sihombing[2], Syahril Efendi[3], Suherman[4]
Student of Doctoral Program in Computer Science[1]
Department of Computer Science-Faculty of Computer Science and Information Technology,
Universitas Sumatera Utara, Medan, Indonesia[1, 2, 3]
Department of Electrical Engineering-Faculty of Engineering, Universitas Sumatera Utara, Medan, Indonesia[4]

***Abstract*—Traditional load balancing systems frequently have trouble adjusting to abrupt and unexpected changes in traffic. This can cause problems like server overload, longer response times, and more requests being denied. This problem is highly important in areas like healthcare, finance, cloud computing, and e-commerce, where performance, stability, and fast data delivery are all very important. To solve this problem, this study presents ReAdaBalancer, an adaptive load balancing architecture that aims to improve system performance, scalability, stability, and efficiency in contexts with changing traffic. Flask serves as the backend framework for ReAdaBalancer, while Nginx serves as the load balancer. Real-time monitoring and analytics are used to improve traffic distribution based on the resources that are currently available. Leveraging queuing theory (M/M/s/K Network), the system's performance is tested under diverse load situations, providing insights into its scalability and efficiency. ReAdaBalancer can also learn and adapt all the time, thanks to machine learning and heuristic optimization. This makes sure that it works the same way even when demand changes. Experimental results demonstrate that, under equivalent settings, ReAdaBalancer decreases response times by over 67% and reduces request denial rates by over 50% in comparison to traditional methods. This work has multiple opportunities for subsequent investigation. Future improvements could involve making ReAdaBalancer work in distributed multi-data center environments, adding reinforcement learning to make decisions more independently, looking into load balancing strategies that use less energy, and making it work in edge computing and IoT ecosystems.**

***Keywords*—Cloud computing; heuristic optimization; adaptive load balancing; scalability; ReAdaBalancer**

I. INTRODUCTION

Cloud computing has gained significant popularity in recent years due to the rapid advancement of information and communication technology, which has driven the growth of data centers with resource virtualization and high connectivity [1]. To handle the increasing complexity of cloud environments, this study proposes the Rock Hyrax approach, an algorithmic model inspired by the adaptive and collaborative behavior of hyrax animals. The approach is designed to optimize load balancing in cloud computing systems, ensuring flexibility, scalability, and continuous access to resources [2]. Efficient load management is essential for maintaining performance while reducing resource waste. The Rock Hyrax model mimics the collective intelligence of hyrax groups, allowing computing systems to adapt dynamically to fluctuating workloads through intelligent collaboration. However, the rapid expansion of cloud services also introduces challenges such as system reliability, latency reduction, and energy efficiency. Efficient cloud computing faces several challenges, including latency, dark data risks, governance issues, data availability, privacy, long-term costs, and potential threats from edge computing and related technologies [3]. One promising solution to address these challenges is the implementation of efficient load-balancing techniques. Among innovative approaches, the Rock Hyrax method stands out by emphasizing adaptive, collective strategies inspired by the social behavior of hyrax herds, which are adept at distributing risks and surviving in dynamic environments.

In cloud computing, load balancing distributes workloads across smaller nodes, enhancing system performance and stability [4, 5]. The Rock Hyrax strategy draws inspiration from the cooperative survival tactics of hyrax animals, which allocate roles and adapt effectively in unstable habitats. Existing load balancing algorithms can be broadly categorized into three groups: static, dynamic, and nature-inspired methods [5]. To address modern networked cloud environments, newer approaches must be smarter, more efficient, and adaptive.

Static Load Balancing (SLB) leverages prior knowledge of system resources such as memory, storage, and processing power, resulting in lower overhead compared to dynamic methods [6, 7]. However, SLB lacks flexibility in adapting to changing workloads. Conversely, Dynamic Load Balancing (DLB) offers higher adaptability by adjusting workloads in real time, but at the cost of increased overhead [7]. Nature-inspired approaches, such as the Rock Hyrax algorithm, integrate the strengths of both models by mimicking decentralized, socially adaptive survival strategies. This enables accurate workload distribution with reduced overhead, even under dynamic and uncertain system conditions.

In contrast, Nature Inspired Load Balancing (NLB) algorithms mimic biological and natural processes such as genetic evolution and honey bee foraging [8]. Compared to traditional methods, NLB is generally more effective in solving highly complex problems by approaching global optima rather than local ones. For example, Round Robin, although widely applied in cloud computing, often struggles with scalability and adaptability in dynamic environments. With the growing number of users, scheduling and load balancing techniques have become essential to sustain the performance of cloud

*Corresponding Author.

networks [9]. In this context, the Rock Hyrax approach offers advantages in social adaptability and efficient load distribution, even under unstable network conditions.

Load balancing also plays a key role in optimizing task assignment to virtual machines according to priority [10]. In static implementations, cluster nodes are connected through fixed routing and exchange specialized information for advanced processing [11]. By contrast, dynamic algorithms monitor traffic conditions in real time and redistribute workloads adaptively. This continuous feedback mechanism improves resource utilization, performance, and response time without relying on prior system information [12, 13].

Nature-inspired methods are becoming increasingly popular in this area. For instance, Particle Swarm Optimization (PSO) simulates the collective behavior of fish schools or bird flocks in search of optimal solutions [5, 6, 14]. Other inspirations include the human immune system, which adaptively detects and isolates threats, and physical phenomena such as earthquakes or avalanches, which inform automated post-failure recovery models [15]. Since evolutionary principles can be applied independently of the underlying medium, they enable the development of highly adaptive cloud systems [16].

This study also highlights the theoretical importance of latency and accessibility in cloud load balancing. By modeling more complex dynamic systems, it becomes possible to maximize workload distribution while considering energy efficiency. Therefore, exploring Rock Hyrax and Regret Minimization approaches is highly relevant, as they can optimize resource allocation, reduce energy consumption, and enhance the operational efficiency of machine learning–based applications, including evolving recommendation systems.

This study is organized as follows: Section I introduces the background and related work, Section II describes the design and methodology of ReAdaBalancer, Section III presents the implementation details and experimental setup and also discusses the results and analysis. Finally, Section IV concludes the study with key findings and directions for future research.

II. Materials and Methods

A. Cloud Computing

Cloud computing has changed the way businesses use and manage their IT resources. Now, they focus on ensuring that resources are used efficiently and the load is balanced to get the best performance. Research conducted a thorough review of cloud computing usage and emphasized how important load balancing is for cloud service performance [17]. Dynamic load balancing approaches are becoming increasingly important as cloud setups get more complicated. This approach helps to make resources more efficient and reduce system overhead. Researchers wrote a long report on dynamic load balancing in cloud computing. They talked about how difficult it is for cloud service providers to keep their services available and reliable while also avoiding resource overload [18]. The research suggested using nature-inspired metaheuristic algorithms to improve load balancing [19]. This shows that nature-inspired methods can be used to find flexible solutions in changing contexts. All these studies underline the importance of using advanced optimization algorithms to solve load distribution problems in cloud computing systems.

Despite advancements in cloud computing load balancing, there is still a big gap in getting the best resource distribution in networked systems, especially when workloads and resource limits are constantly changing [19]. Current methods frequently have trouble dealing with the intricacies of networked cloud infrastructures, where the dependency of several nodes and changing traffic conditions make load balancing difficult. This study suggests that combining the Rock Hyrax strategy with Regret Minimization strategies could lead to a better solution by reducing the regrets people have about their decisions in real-time load-balancing situations [22]. The goal of this study is to create a new optimization framework that can adjust to the changing nature of cloud networks. This will make better use of resources and improve the overall performance of the system [20, 21]. However, it still leaves room for further exploration, especially in quantifying the impact of factors such as technological readiness, top management support, relative advantage, competitive pressure, organizational resistance, system complexity, and data security [23, 24, 31]. Table I shows a critical literature analysis.

TABLE I. Critical Literature Analysis

Study /Approach	Contribution	Limitation / Gap Identified
Dynamic Load Balancing [17,18]	Improves efficiency, reduces overhead	Struggles under highly dynamic workloads and distributed nodes
Nature-inspired Metaheuristics [19]	Flexible, adaptive solutions	Often computationally expensive, lacks real-time responsiveness
Regret Minimization [22]	Handles decision-making under uncertainty	Not yet applied in large-scale cloud balancing contexts
Cloud Performance Factors [23,24,31]	Identify organizational & technical influences (e.g., security, complexity)	Mostly conceptual, not integrated into optimization algorithms

B. RHSO

Many different methods have been created to make the most of energy use in cloud computing systems, but a lot of them still have trouble managing load distribution well in a changing and unpredictable environment [32]. Researchers have come up with hybrid ways to use less energy, but they don't completely solve the problems of uncertainty and load changes in cloud computing [25]. When working with data that changes a lot and is hard to understand, like loads that change or are hard to estimate, these methods can run into problems. This shows that only optimizing energy isn't enough to get the job done without taking into account how the load changes in the system. Also, the Rock Hyrax Swarm Optimization (RHSO) algorithm that was created works well for selecting features for credit card fraud detection systems, but it is still not very useful in the cloud computing world [26]. These methods are more often used for classification and pattern recognition tasks, and they don't always take into account how uncertain it is to manage resources in the cloud. We still need to look into how well RHSO can optimize load distribution in real time in a cloud computing environment with different resources and needs that aren't always clear [27, 28]. There are many different load optimization models, such the Random Regret Minimization (RRM) model. However, it still has problems when it comes to

being used in the cloud computing environment, which is very dynamic and uncertain [29]. On the other hand, Bayesian Optimization with expectation correction for cumulative regret reduction and other similar methods are more focused on finding the best solution in a noisy environment [30].

This study's goal is to fill in the gaps by creating a new method that uses both the Rock Hyrax Swarm Optimization algorithm and the Regret Minimization strategy to optimize load balancing in cloud computing networks. One of the main goals of this research is to find a solution to the problem of uncertainty in load distribution that cloud computing systems confront. This is because they often need to allocate resources fast and accurately when demand changes. The system can lower losses from bad judgments, make load sharing more efficient, and use less energy more effectively by utilizing a Regret Minimization strategy based on modularity optimization [13]. This study focuses to improve the Rock Hyrax and Regret Minimization methods for optimizing load balancing on networked cloud-based computing systems. The goal of this study is to find and improve optimization methods that use algorithms to make cloud computing load allocation decisions more accurate. One of the problems with load-balancing is how to deal with changing and different loads without affecting performance or resource availability. The idea behind this study is that using Rock Hyrax with Regret Minimization will give better results than traditional load-balancing approaches.

C. Research Stages

The focus of this research is to identify problems associated with throughput optimization to produce an optimal transfer rate of data delivery services. The stages used in this research are developed from previous research. Making of the modelling algorithm is given in the following Fig. 1:

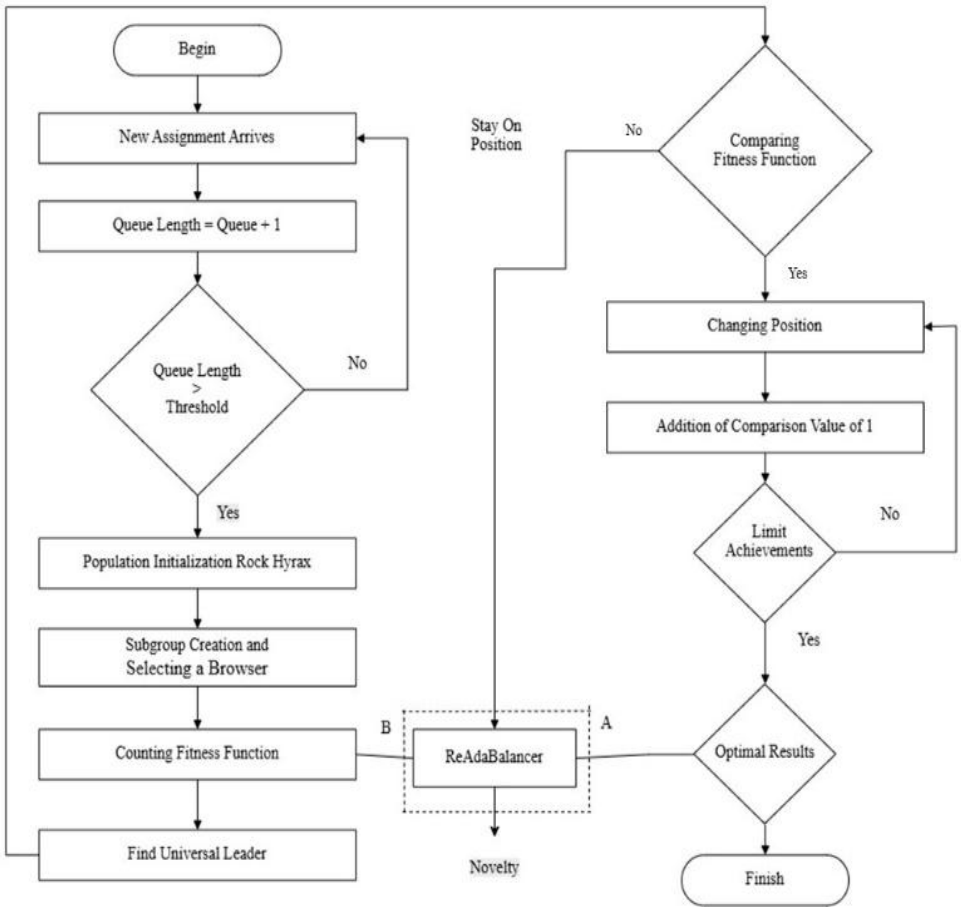

Fig. 1. Process algorithm.

The Proposed Rock Hyrax Load Balancing Algorithm is intended to efficiently manage workloads in a cloud computing environment. The following is a complete process flow story of how the algorithm functions:

1) Job submission: Users submit their tasks to the broker, which serves as the central controller for job scheduling and resource allocation in the cloud environment. The broker is also responsible for managing the distribution of tasks across different virtual machines (VMs).

2) Initial load assessment: The algorithm starts by checking the workload and resources of each virtual machine to find virtual machines that are overloaded. This is done by comparing the current load with a preset threshold.

3) Threshold monitoring: The waiting job queue length for each virtual machine (VM) has a threshold. If the queue length exceeds this threshold, the algorithm triggers load-balancing actions to ensure that no single VM becomes a bottleneck due to excessive demand.

4) Dynamic load redistribution: The algorithm evaluates the current state of the virtual machine (VM) when new jobs arrive. If the VM is overloaded, the algorithm reallocates waiting tasks to other VMs with available capacity. Dynamic redistribution and monitoring ensure optimal performance and prevent server failure.

5) Quality of Service (QoS) considerations: The Rock Hyrax algorithm considers quality of service parameters such as maxspan, throughput, response time, and energy efficiency. This ensures that load balancing addresses live workloads and optimizes overall system performance.

6) Continuous feedback loop: The capacity and workload of each server are monitored and sent back to the broker. The broker can make informed decisions about task allocation and adjust load distribution strategies as needed with this feedback loop.

7) Execution and monitoring: Once tasks are allocated to the appropriate virtual machines (VMs), the algorithm continuously monitors the execution of these tasks. If any VM is overloaded again, the process repeats, ensuring that the system remains balanced and efficient during job execution.

Performance Evaluation: Finally, the results show that makespan and energy consumption become lower, which indicates that Rock Hyrax's load balancing algorithm really works well when organizing resource allocation in cloud computing.

D. Preprocessing and Augmentation Data

This study looks at common preprocessing or data augmentation methods that are utilized in machine learning. If we think about the process in terms of data handling and optimization, though, we can see that the system is doing some "preprocessing" processes to manage tasks. For instance, when a new task comes in, the system will add it to the queue and check to see whether it goes over the limit. This is like getting data ready for more processing. This phase makes sure that only jobs that are too heavy are processed, which cuts down on unneeded overhead. Also, setting up the "Rock Hyrax" population and making subgroups can be considered as setting up several possible solutions (or data points) for optimization, like feature selection or data splitting. The system does "data augmentation" by comparing fitness functions and changing the placements of agents based on these comparisons. This is done again and over again. You could say that this is an addition to the original set of possible answers, with each repetition making the solutions better and more suited to the problem at hand. Also, the addition of a comparison value and the constant changes to the agents' positions are like the iterative nature of data augmentation, where new data variations are made by making changes and comparisons, which makes the model better at adapting and finding the best solution. These phases make sure that the optimization process looks at a lot of different options, which makes the system stronger and more efficient, just like how data augmentation makes models more general.

E. Model Architecture

The model architecture refers to the overall structure of a machine learning model or computational model, which defines the components, their relationships, and the flow of data through the system. The model architecture serves to design, build, and understand how the various parts of the model interact to achieve the desired functionality.

The following is an architectural diagram of the M/M/s/k queue pseudocode based on the pseudocode that has been made. This diagram illustrates how customers enter the system, wait in the queue if needed, are served by the server, and then exit the system or are rejected if the queue is full, as can be seen from Fig. 2:

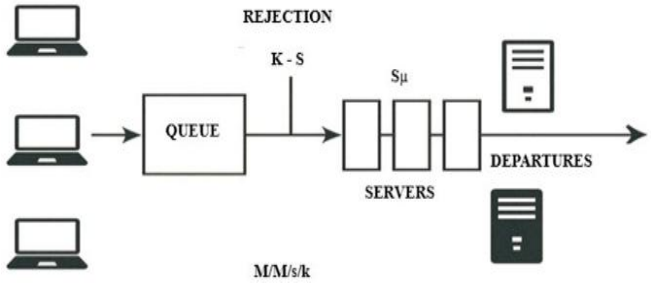

Fig. 2. Architectural diagram of the M/M/s/k queuing model.

Fig. 2 shows a queuing system in a crowded place, such as a fast food restaurant or a call centre. Randomly come to the queue of customers (represented by a computer icon). However, this queue has a limit that not everyone can join. If the queue is full (exceeding k capacity), you will be denied entry. In this system, several servers (s) serve customers one by one, and each server has a certain service speed (μ). Once served, exit the system. This model is called M/M/s/k, and is often used to measure the performance of capacity-constrained systems, such as how fast the service is, and how busy the system itself is.

Load balancers distribute load based on server capacity and user demand. Basic formula:

$$S_i = argarg\left(C\left(S_j\right) + L\left(S_j\right)\right), \quad (1)$$

where, *Si* is the selected server;

Sj is each server in the pool;

C(*Sj*) is the processing capacity of server j;

L(*Sj*) is the latency of server j.

The load balancer selects the server with the lowest C+L value.

1) ReAdaBalancer algorithm: ReaAda balances tasks with the smallest load and considers redistribution. Determining the smallest load on a server can be analogous to a mathematical formula that describes the probability of a server being active or not, based on the load received. With a value, which is the size of the load, and other parameters, such as α and β, which serve to adjust the sensitivity and threshold. Creating the server inactive is significantly larger, creating an efficient that when the server does not receive many requests.

$$P(S_i) = \frac{1}{1 + e^{-(\alpha U_i - \beta)}}, \quad (2)$$

where, $P(S_i)$ is the probability of selecting server i;

U_i is the utilization of server i;

α and β are parameters that control the sensitivity of the algorithm to server load. The lower the U_i value, the greater the chance of that server being selected.

2) Regret minimization: Regret Minimization aims to minimize the difference between the best decision and the decision taken:

$$R_t = \sum_{t=1}^{T} w_t (O_t - A_t), \quad (3)$$

where, R_t is the regret at time t;

O_t is the optimal outcome that should be obtained if the best decision is taken;

A_t is the result obtained from the decision taken by the system;

W_t is the weight that determines how important the error is at time t.

The goal of this algorithm is to make $R_t \rightarrow 0$ over time by adaptive learning.

3) Performance evaluation: Performance evaluation is calculated by looking at throughput (T), response time (R), and error rate (E):

$$K = \frac{T}{R + E}, \quad (4)$$

where, K is the performance score;

T is the number of successfully processed requests;

R is the average response time;

E is the number of errors that occur.

The higher the K, the better the system performance.

Based on the idea of queuing theory, the M/M/k Load Balancer Architecture is arranged here in an organized table. Typically used in high-performance distributed systems where multiple servers (k) handle randomly arriving jobs (Poisson process) and each server offers exponential service time. Describes the architectural structure of the M/M/k queue system with a Load Balancer. These system requests arrive at a rate of λ and are then distributed by the Load Balancer to the k available servers. If all servers are busy, they enter the Queue before being processed. Utilization (ρ) is an important parameter that determines whether or not the system is in a stable state. There are two main probabilities: P_0, which indicates that there are no customers in the system, and Pq, which indicates the chance that customers will have to queue before being served.

Output based on system performance is rather different. While L (System Length) shows the total customers in the system, including those being served. Shows L (Queue Length). In terms of time, Wq (Queue Wait Times) calculates how long the average customer has to wait in the queue, while W (System Wait Times) is the total time spent by the system. After proposals, requests go out as completed requests.

F. Simulation Model

A simulation model is a computational representation of a process, system used to simulate under various conditions. This model is built into a diagrammatic architecture.

Here is a diagram of the queue simulation architecture based on the pseudocode you provided. This diagram shows the relationship between the main components, such as customer, simEventList, queue, and queueSim, which can be seen from Fig. 3.

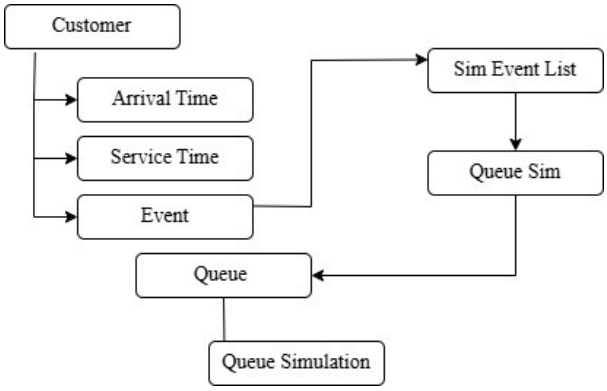

Fig. 3. Architectural diagram of the queue simulation model.

Performance evaluation at the final stage enables continuous monitoring and improvement of the system to remain optimum in handling demand. This approach is not operational, but it also minimizes the potential for errors and quality degradation.

G. Experimental Setup

Experimental setup refers to the arrangement of equipment, in terms of both hardware and software, to analyze the phenomenon under study.

- The hardware and software used for this study are:
  - The research environment (CloudSim, iFogSim);
  - Infrastructure for both private and public clouds (OpenStack, AWS, Azure, GCP);
  - Microservices and container platforms like Kubernetes and Docker;
  - systems for edge and fog computing;
- Frameworks for serverless computing.

Adaptive and decentralized load optimization is its best feature. This makes it very useful in multi cloud, hybrid, and edge-cloud environments where real-time decision-making and self-learning are very important.

III. Results and Discussion

This study show that the integration of Rock Hyrax and

Regret Minimization for cloud computing optimization with the ReAdaBalancer concept keeps the system stable and getting data quickly is very important. This is done by optimizing resource allocation. Under the same conditions, ReAdaBalancer cuts the response time by about 67% and the request rejection rate by more than 50% compared to the standard approach. This is done by adjusting to the traffic in real time and considering the server status. Unlike previous works such as [22], where only using the Rock Hyrax algorithm shows lower accuracy due to its inability to stress the system remains stable in optimizing resource allocation. In contrast to [28], where they propose a near-optimal discrete optimization approach for experimental design using a Regret Minimization framework. This approach aims to select the statistically most efficient subset of design points from a larger data set. In addition, while [29] this RRM model can be adapted to plan resource allocation by considering regret over previous decisions.

A. *Maths Pseudocode*

Between mathematical conceptual thinking and effective programming implementation, Maths Pseudocode is a logical and methodical presentation of mathematical algorithms organized in half-code form. including the following:

#Mathematical pseudocode

import maths

def lambdaN(lam, k, n):

Fig. 4 is the pseudocode of the simulation results using Python.

```
if n < k:
return lam
return 0
def muN(mu, s, n):
if n < s:
return n*mu
return s*mu
def P0(lam, mu, s, k):
sum = 1.0
for i in range(1, s):
sum += (pow((lam/mu), i) / math.factorial(i))
sum  += ((pow((lam/mu),  s)  /  math.factorial(s)))  *  ((1-pow((lam/(s*mu)), (k-
s+1)))) / (1-(lam/(s*mu)))))
 return 1/sum
```

Fig. 4. Arrival function, service level, and probability.

Summary of Fig. 4

The code above implements the M/M/s/k queuing system, which analyses a system with a limited number of servers (s) and a maximum capacity (k). The lambdaN function stops if the capacity is full, which stops the arrival of customers (λ). Calculates the service level (μ), which increases with the number of customers but is limited by the number of servers. The P0 function terminates an empty system (P_0), which is useful for evaluating queuing systems. This model is useful for others with capacity constraints, such as call centers, computer servers, or service systems.

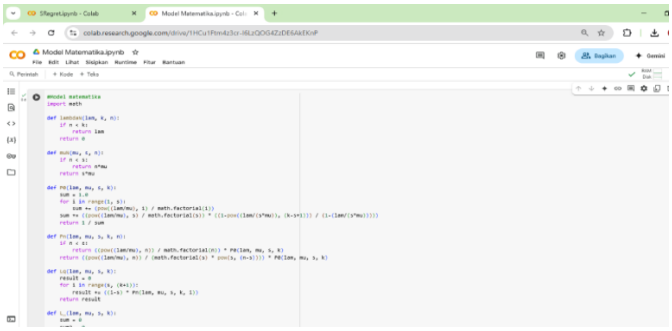

Fig. 5. Running a mathematical model.

Fig. 5 shows a snippet of Python code written on Google Colab, titled Mathematical Models. This script implements several complex mathematical functions used for probabilistic modelling or queuing theory, marked with common notations such as λ (arrival rate), μ (service rate), and k (number of customers in the system). The functions P0 and Pn indicate attempts to calculate the probability of states in a queuing system, while the functions PC and PQ indicate probability calculations in the context of a limited system capacity or queue. The use of math.factorial, math.pow, and the use of nested loops reflect typical combinatorial and exponential calculations in the Poisson or M/M/1/K models.

B. *Simulation Pseudocode*

Run through simulations, before they are used in actual computer code, pseudocode, a methodical description of a logically ordered simulation process, allows the systematic construction and analysis of scenarios, therefore modelling the dynamics of complex systems and enabling the comprised. The following Fig. 6 shows the customer function, simEventList..

```
#Simulation pseudecode
import numpy as np
import random
import maths
import queue
class customer:
  departTime = 0
  def __init__(self, at, st, s):
        self.arrivalTime = at
        self.serviceTime = st
        self.event = s
class simEventList:
  def __init__(self, lam, mu, n = 0):
        tempServiceTime = int(np.random.exponential(1/mu)*60)
        self.list = [customer(0, tempServiceTime, 'arrival')]
        for i in range(n-1):
              tempServiceTime = int(np.random.exponential(1/mu)*60)
              tempInterArrivalTime = int(np.random.exponential(1/lam)*60)
```

Fig. 6. Customer function, simEventList .

Summary of Fig. 6

Part of a queuing system simulation that uses an exponential distribution to determine customer arrival and service times. Customer The class represents a customer with attributes of arrival time, service time, and event type (or departure). Meanwhile, the simEventList class manages the list of events in the simulation, generates service times, and determines the inter-arrival time of customers based on an exponential

distribution. This simulation is useful for analyzing the performance of other systems that have limited capacity, with queues, such as contact centers, banks, or other systems.

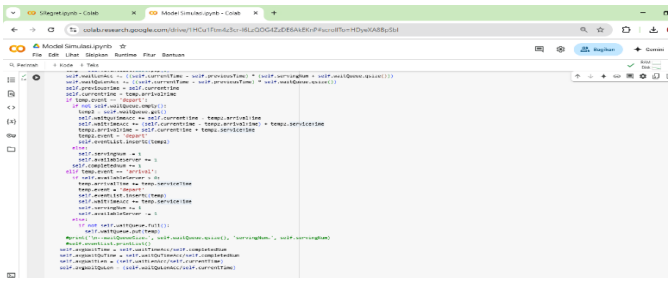

Fig. 7. Running system simulation.

Summary of Fig. 7

Simulation of a queuing system that uses an exponential distribution to determine customer arrival and service times. Each customer has attributes of arrivalTime, serviceTime, and event type. In this simulation, events are in the simEventList class, which ensures that each customer's arrival and departure are sorted chronologically. The available servers, as well as calculating performance metrics such as average time and queue length, the queueSim class is responsible for running the simulation by maintaining the queue. If all servers are busy, customers will be put into a waiting queue, provided there is still available capacity.

The simulation is run in a loop that processes each event in the list sequentially. If a customer arrives and there is a server, they are served immediately. Otherwise, they enter the queue if there is still space. Once in the queue, they begin to be served, the server becomes available again, and the next customer follows. The final results of the simulation include various statistical measures that can be used to evaluate the efficiency of the queuing system, such as average waiting time and queue length. The model can be used in various real-world applications, such as customer service, traffic management, and data processing systems.

C. Load Balancer System Test Pseudocode

The Load Balancer System Test Pseudocode is designed to confirm the fair and efficient traffic distribution between servers by logical simulation before technical implementation is carried out. Pseudocode is a methodical depiction of the load balancer system test flow comprising the following, as shown in Fig. 8.

Summary of Fig. 8

The load balancing () function simulates the M/M/k queue model with multiple servers. It calculates system utilization, estimated number of customers in the system and queue and arrival rate (λ), service rate (μ), and arrival rate (λ). With this simulation, it is possible to evaluate service efficiency in various queuing systems, such as call centres, hospitals, and data centres, in order to optimize capacity and reduce customer waiting time.

Fig. 9 shows the running of simulation results using Python.

```
import numpy as np
import matplotlib.pyplot as plt
def loadBalancer(lam, mu, k, s=5):
 """
Simple simulation of the M/M/k queue model.
lam: arrival rate (lambda)
mu: service rate (mu)
k: number of servers
s: number of lambda simulation points
Returns:
x -> lambda array
simLy, Ly -> number of customers in the system (simulation vs theory)
simWy, Wy -> customer waiting time in the system (simulation vs theory
simLqy, Lqy -> number of customers in queue (simulation vs theory
simWqy, Wqy -> waiting time in queue (simulation vs theory)
 """
x = np.linspace(lam, lam + 1, s) # Simulating multiple lambda values
rho = x / (k * mu) # System utilisation
# M/M/k queue system estimation formula
Ly = rho / (1 - rho) # Estimated number of customers in the system
Wy = Ly/x # Estimated total waiting time
Lqy = Ly - (x/mu) # Estimated number of customers in the queue
Wqy = Lqy / x # Estimated waiting time in queue
```

Fig. 8. Load balancer function.

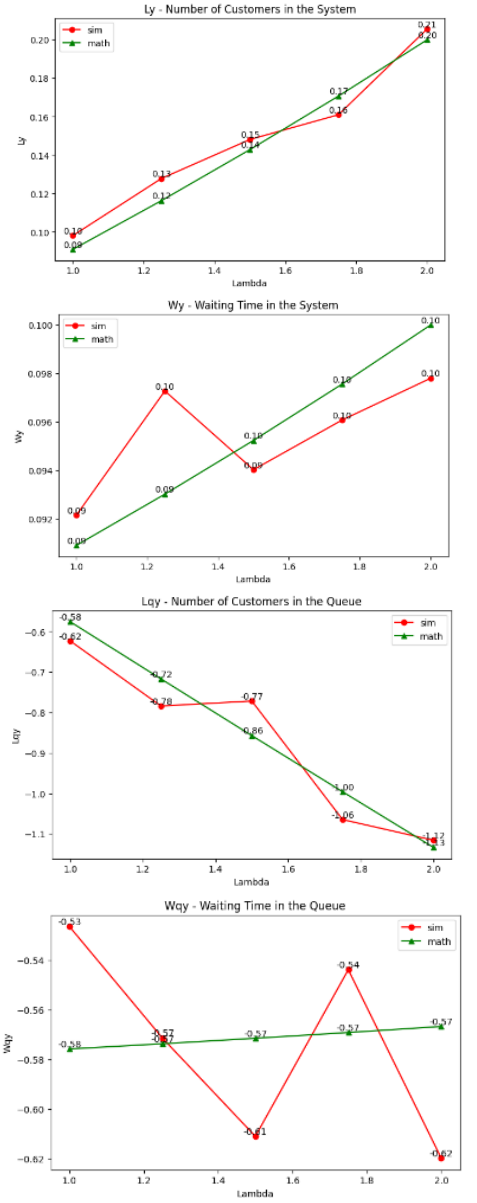

Fig. 9. Running simulation test of load balancer system.

Simulation summary of Load Balancer Test Model

Load balancing, with the code above, simulates a multiple server M/M/k queue system. The load balancing function calculates queue parameters such as the number of customers in the system (Ly), the waiting time of customers in the system (Wy), the number of customers in the queue (Lqy), and the waiting time in the queue (Wqy). These results are calculated both theoretically and through simulations of varying the value of λ (arrival rate) within a certain range. The simulation is added with random noise in the real system to represent in. After the calculation is complete, the results are visualized using Hutmatplotlib to compare the theoretical values and simulation results.

However, there are some implementations of this. Ly is not suitable for calculating Ly due to multiple servers (M/M/k), as it only applies to M/M/1 systems. Not checking the system, which should ensure that the utilization ratio $\rho = \lambda/(k\mu)1$ in order to keep the queue under control. If $\rho > 1$, the system is not able to increase indefinitely. For improvement, it is necessary to use the Erlang-C formula to get a more accurate estimation in the M/M/k model.

D. Innovative ReaAdaBalancer

An innovative load balancing method called RegretAdaptive Load Balancing (ReAdaBalancer) dynamically optimizes demand distribution by means of regret reduction. The simulation in Table II shows how helpful this approach is in settings with varying workloads and heterogeneous servers.

TABLE II. REGRETADAPTIVE LOAD BALANCER PERFORMANCE RESULTS

Server ID	Final Load	Final Regret	Utilization (%)
0	11	0	11.0%
1	5	0	5.0%
2	0	0	0.0%
3	0	0	0.0%
4	6	0	6.0%

Table Analysis:

1) Final Load: Shows the final load amount that each server has after 20 iterations.

2) Final Regret: None of the servers are overloaded, so the regret score remains 0.

3) Utilization (%): All servers have a low utilization rate, which means the load has been distributed fairly well.

Fig. 10 shows the running of simulation results using Python.

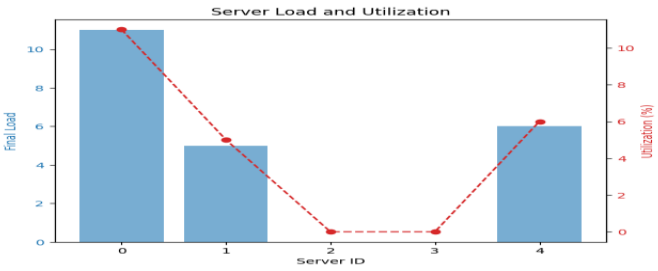

Fig. 10. Simulation graph of adaptive load balancer regret.

This graph presents "Server Load and Utilization" data related to the final load and utilization rate of five server units identified by IDs 0 to 4. The observation shows that there is a significant imbalance in the workload distribution between servers. The server with ID 0 had the highest final load of 11 and a utilization rate of 10%, indicating that it was working intensively and was likely to be the main center of data processing. Meanwhile, server ID 1 showed a drastic drop with an end load of only 5 and a utilization rate of 5%, suggesting a reduction in activity or an imbalance in load allocation. More strikingly, server IDs 2 and 3 recorded zero values on both metrics, signifying that they were not involved in any computing processes at all, potentially indicating high idle time or no assignment at all. In contrast, server ID 4 shows an increase again with a final load of 6 and utilization of 6 %, indicating moderate involvement in system activity. Overall, this graph reflects an uneven load distribution among the server units. This imbalance can have an impact on operational efficiency as well as hardware lifespan. Therefore, there is a need to evaluate the scheduling and load distribution strategies to ensure that all system resources are utilized optimally and proportionally.

E. Discussions

In summary, the discussion confirmed that the proposed ReAdaBalancer, which includes preprocessing as an important first step to prepare the data for the load balancing [10] process, and augmentation, which improves the model in the face of dynamic changes in cloud environments, both help provide smart and efficient solutions for load optimization that can be used on a large scale and in cloud environments that change quickly.

Despite its advantages, this research has some limitations, including that the algorithm becomes more difficult to compute as the number of nodes and the size of the cloud network increase. The algorithm may have difficulty in making timely adjustments when there is a sudden change in load or system failure, which can lead to poor conclusions. Future improvements may include: 1) the use of parallel computing and distributed processing to speed up the performance of the algorithm, 2) the use of online learning or incremental learning techniques to keep the model up-to-date as new data comes in, 3) the addition of reinforcement learning for parameter adjustment to provide a dynamic solution that avoids incorrect parameter selection, 4) the addition of distributed error detection and self-healing features to the algorithm can make it more fault-tolerant. For example, replication techniques can be used to copy the load to a backup server in case of a failure, and 5) the use of hybrid algorithms that combine Rock Hyrax with other faster optimization methods such as shortest path algorithms or approximate algorithms can help you make decisions faster and ensure that load distribution occurs quickly and efficiently for real-time applications.

ReAdaBalancer has been shown to work very well for improving system resilience and availability. Helping to make better use of resources and adapt to changing conditions. The Rock Hyrax Approach and Regret Minimization will focus on making cloud computing [13] more scalable, able to adapt automatically, able to handle failures, better at managing energy, and more secure in the future. This research can make

a cloud system that is more efficient, safe, and able to adapt to future issues by looking into how to combine emerging technologies like machine learning, blockchain, and real time processing and coming up with solutions for apps that need to do more than one thing.

IV. Conclusion

The model that combines Rock Hyrax and Regret Minimization for cloud computing optimization with the ReAdaBalancer concept keeps the system stable, and the ReAdaBalancer cuts response time by about 67% and request rejection rate by more than 50% compared to the standard approach by adjusting traffic in real-time.

Test Server with ID 0 has the largest final load of 11 and utilization rate of 10%, which means that it is working hard and is probably the major place where data is processed. Server ID 1, on the other hand, has a final load of only 5 and a utilization rate of 5%, which means that either activity has decreased or load allocation has become unbalanced. This performance boost can be a clever and effective way to optimize load that works on a large scale and in cloud environments that change quickly.

Results are in line with the goal and unique features. The suggested load balancing optimization method effectively solves the main problems found in the study, such as being able to adapt to changing system conditions, using a lot of energy, and not being very good at distributing workloads in real time. Rock Hyrax and Regret Minimization work together to give a unique advantage over standard algorithms by changing to traffic patterns in real-time. This makes the system more responsive and uses resources better.

References

[1] Bello, Sururah A., et al. "Cloud computing in construction industry: Use cases, benefits and challenges." Automation in Construction 122 (2021): 103441. Available at: https://doi.org/10.1016/j.autcon.2020.103441.

[2] Shukur, H. et al. (2020) ‘Cloud Computing Virtualization of Resources Allocation for Distributed Systems’, Journal of Applied Science and Technology Trends, 1(2), pp. 98–105. Available at: https://doi.org/10.38094/jastt1331.

[3] Kirtirajsinh Zala et al., (2022), PRMS: Design and Development of Patients' E-Healthcare Records Management System for Privacy Preservation in Third Party Cloud Platforms, https://doi.org/ 10.1109/ACCESS.2022.3198094: IEEE Access. (2022): 85777 - 85791.

[4] Khare, Shivangi, Uday Chourasia, and Anjna Jayant Deen. "Load balancing in cloud computing." Proceedings of the International Conference on Cognitive and Intelligent Computing: ICCIC 2021, Volume 1. Singapore: Springer Nature Singapore, (2022): 978-981. Available at: https://doi.org/10.1007/978-981-19-2350-0_58.

[5] Shafiq, Dalia Abdulkareem, N. Z. Jhanjhi, and Azween Abdullah. "Load balancing techniques in cloud computing environment: A review." Journal of King Saud University-Computer and Information Sciences 34.7 (2022): 3910-3933. Available at: https://doi.org/10.1016/j.jksuci.2021.02.007.

[6] Mishra, Sambit Kumar, Bibhudatta Sahoo, and Priti Paramita Parida. "Load balancing in cloud computing: a big picture." Journal of King Saud University-Computer and Information Sciences 32.2 (2020): 149-158. Available at: https://doi.org/10.1016/j.jksuci.2018.01.003.

[7] Jena, Uttam Kumar, Pradipta Kumar Das, and Manas Ranjan Kabat. "Hybridisation of meta-heuristic algorithm for load balancing in cloud computing environment." Journal of King Saud University-Computer and Information Sciences 34.6 (2022): 2332-2342. Available at: https://doi.org/10.1016/j.jksuci.2021.02.004.

[8] Singh, Abhilash, Sandeep Sharma, and Jitendra Singh. "Nature-inspired algorithms for wireless sensor networks: A comprehensive survey." Computer Science Review 39 (2021): 100342. Available at: https://doi.org/10.1016/j.cosrev.2020.100342.

[9] Katangur, Ajay, Somasheker Akkaladevi, and Sadiskumar Vivekanandhan. "Priority weighted round robin algorithm for load balancing in the cloud." 2022 IEEE 7th international conference on smart cloud (SmartCloud). IEEE, (2022): 5179. Available at: https://doi.org/10.1109/SmartCloud57250.2022.00104.

[10] Jaganathan, Subash Chandra Bose, Ramesh Saha, and S. Kannadhasan. "An Efficient Enhanced Dynamic Load Balancing Weighted Round Robin Algorithm for Virtual Machine in Cloud Computing." Journal of Algebraic Statistics 13.2 (2022): 2121-2128. Available at: https://doi.org/10.17762/jas.v13i2.394.

[11] Nakas, Christos, Dionisis Kandris, and Georgios Visvardis. "Energy efficient routing in wireless sensor networks: A comprehensive survey." Algorithms 13.3 (2020): 72. Available at: https://doi.org/10.3390/a13030072.

[12] Kashani, Mostafa Haghi, and Ebrahim Mahdipour. "Load balancing algorithms in fog computing." IEEE Transactions on Services Computing 16.2 (2022): 1505-1521.Available at: https://doi.org/10.1109/TSC.2022.3174475.

[13] Pradhan, Arabinda, Sukant Kishoro Bisoy, and Amardeep Das. "A survey on PSO based meta-heuristic scheduling mechanism in cloud computing environment." Journal of King Saud University-Computer and Information Sciences 34.8 (2022): 4888-4901. Available at: https://doi.org/10.1016/j.jksuci.2021.01.003.

[14] Akhtar, Talha, Najmi Ghani Haider, and Shariq Mahmood Khan. "A comparative study of the application of glowworm swarm optimisation algorithm with other nature-inspired algorithms in the network load balancing problem." Engineering, Technology & Applied Science Research 12.4 (2022): 8777-8784. Available at: https://doi.org/10.48084/etasr.4999.

[15] Breish, Firas, Christian Hamm, and Simone Andresen. "Nature's Load-Bearing Design Principles and Their Application in Engineering: A Review." Biomimetics 9.9 (2024): 545. Avalable at: https://doi.org/10.3390/biomimetics9090545.

[16] Jiao, Licheng, et al. "Nature-Inspired Intelligent Computing: A Comprehensive Survey." Research 7 (2024): 1-38. Available at: https://doi.org/10.34133/research.0442.

[17] Maina Lawan, M., Oduoza, C. and Buckley, K. (2021) ‘A Systematic Review of Cloud Computing Adoption by Organisations’, International Journal of Industrial and Manufacturing Systems Engineering, 6(3), p. 39. Available at: https://doi.org/10.11648/j.ijimse.20210603.11.

[18] Sharmah, D. and Bora, K.C. (2024) ‘A Survey on Dynamic Load Balancing Techniques in Cloud Computing’, Lecture Notes in Electrical Engineering, 1088(2), pp. 273–282. Available at: https://doi.org/10.1007/978-981-99-6855-8_21.

[19] Gupta, R. and Sharma, O.P. (2024) ‘Optimization of Load Balancing in Cloud Computing through Nature- Inspired Metaheuristic Algorithms’, pp. 3216–3226. Available at: https://doi.org/10.52783/jes.8054.

[20] Sakhri, A. et al. (2024) ‘A digital twin-based energy-efficient wireless multimedia sensor network for waterbirds monitoring’, Future Generation Computer Systems, 155(February), pp. 146–163. Available at: https://doi.org/10.1016/j.future.2024.02.011.

[21] Chawla, K. (2024) ‘Reinforcement Learning-Based Adaptive Load Balancing for Dynamic Cloud Environments’. Available at: http://arxiv.org/abs/2409.04896.

[22] Singhal, S. et al. (2024) ‘Energy Efficient Load Balancing Algorithm for Cloud Computing Using Rock Hyrax Optimization’, IEEE Access, 12(February), pp. 48737–48749. Available at: https://doi.org/10.1109/ACCESS.2024.3380159.

[23] Singhal, S. et al. (2023) ‘Energy Aware Load Balancing Framework for Smart Grid Using Cloud and Fog Computing’, Sensors, 23(7). Available at: https://doi.org/10.3390/s23073488.

[24] Yang, P. et al. (no date) ‘Reducing Idleness in Financial Cloud Services via Multi-objective Evolutionary Reinforcement Learning based Load Balancer arXiv : 2305. 03463v2 [cs .NE] 23 Nov 2023’. Available at: https://doi.org/10.1007/s11432-023-3895-3.

[25] Kak, S. M., Agarwal, P., Alam, M. A., & Siddiqui, F. (2024). A hybridized approach for minimizing energy in cloud computing. Cluster Computing, 27(1), 53-70. Available at: https://doi.org/10.1007/s10586-022-03807-9.

[26] Padhi, Bharat Kumar, et al. "RHSOFS: feature selection using the rock hyrax swarm optimisation algorithm for credit card fraud detection system." Sensors 22.23 (2022): 9321. Available at: https://doi.org/10.3390/s22239321.

[27] Jagadesh, B. N., et al. "Segmentation Using the IC2T Model and Classification of Diabetic Retinopathy Using the Rock Hyrax Swarm-Based Coordination Attention Mechanism." IEEE Access (2023): 124441 - 124458. Availavle at: https://doi.org/10.1109/ACCESS.2023.3330436.

[28] Allen-Zhu, Zeyuan, et al. "Near-optimal discrete optimisation for experimental design: A regret Minimization approach." Mathematical Programming 186 (2021): 439-478. Available at: https://doi.org/10.1007/s10107-019-01464-2.

[29] Mengjie L et al., Random Regret Minimization Model for Variable Destination- Oriented Path Planning, https://doi.org/ 10.1109/ACCESS.2020.3021524. IEEE Access (2020). 163646 - 163659.

[30] Hu, Shouri, et al. "Adjusted expected improvement for cumulative regret Minimization in noisy bayesian optimisation." arXiv preprint arXiv:2205.04901 (2022):1:31. Available at: https://doi.org/10.48550/arXiv.2205.04901.

[31] Xia, J., Li, M., & Guo, Y. (2009). The Queuing Model of M/M/S/K+ M Based on the Impatience and Changeable Service Rate. Journal of Systems Science & Information, 7(3). Available at: https://doi.org/10.1007/s10586-022-03807-9.

[32] Cui, T., Yang, R., Fang, C., & Yu, S. (2023). Deep reinforcement learning-based resource allocation for content distribution in IoT-edge-cloud computing environments. Symmetry, 15(1), 217. Available at: https://doi.org/10.3390/sym15010217.