Received 25 February 2024, accepted 10 March 2024, date of publication 21 March 2024, date of current version 9 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3380159

RESEARCH ARTICLE

Energy Efficient Load Balancing Algorithm for Cloud Computing Using Rock Hyrax Optimization

SAURABH SINGHAL[1], (Member, IEEE), ASHISH SHARMA[1], ANUSHREE[1], PAWAN KUMAR VERMA[2], MOHIT KUMAR[3], (Member, IEEE), SAHIL VERMA[4,5], (Senior Member, IEEE), KAVITA[6], (Senior Member, IEEE), MANINDER KAUR[7], JOEL J. P. C. RODRIGUES[8], (Fellow, IEEE), RUBA ABU KHURMA[9], AND MARIBEL GARCÍA-ARENAS[10,11]

[1]Department of Computer Engineering & Applications, GLA University, Uttar Pradesh 281406, India
[2]Department of Computer Science & Engineering, Sharda University, Uttar Pradesh 201310, India
[3]Department of Information Technology, MIT Art, Design & Technology University, Maharashtra 412201, India
[4]Manipal University, Rajasthan 303007, India
[5]Universidade Federal do Piauí, Teresina, Piauí 64049-550, Brazil
[6]Uttaranchal University, Dehradun, Uttarakhand 248007, India
[7]Guru Gobind Singh College for Women, Chandigarh 160019, India
[8]Amazonas State University, Manaus, Amazonas 69060-001, Brazil
[9]MEU Research Unit, Faculty of Information Technology, Middle East University & Applied Science Research Center, Applied Science Private University, Amman 11831, Jordan
[10]Department of Computer Engineering, Automatics and Robotics, University of Granada, 18012 Granada, Spain
[11]Centro de Investigación en Tecnologías de la información y las Comunicaciones de la Universidad de Granada (CITIC-UGR), 18071 Granada, Spain

Corresponding author: Maribel García-Arenas (mgarenas@ugr.es)

This work is part of the Grant C-ING-027-UGR23 funded by la Consejería de Universidad, Investigación e Innovación and by European Regional Development Fund (ERDF) Andalusia Program 2021-2027; in part by Ministerio Español de Ciencia e Innovación funded by MICIU/AEI/10.13039/501100011033 under Projects PID2020-115570GB-C22 and PID2022-137461NB-C31 and finally in part by Cátedra de Empresa Tecnología para las Personas (UGR-Fujitsu).

ABSTRACT Cloud computing offers dynamic, scalable, and virtualized computing resources to end users over the internet. Load balancing is crucial for efficient resource use, distributing workloads across multiple resources to prevent overloading. Load balancing is crucial for resource utilization and processing time reduction, but traditional algorithms are often stuck at local maxima, leading to unequal allocation and performance decline. A metaheuristic based algorithm is proposed to dynamically adjust load distribution, ensuring resilience and sensitivity to changing workloads while managing energy consumption. This research presents a Rock Hyrax-based load balancing algorithm that addresses local maxima and power efficiency issues using QoS parameters. The algorithm's performance is evaluated qualitatively and statistically, considering both static and dynamic modes of jobs and virtual machines. Comparing it with existing scheduling algorithms, the algorithm reduces makespan by 10%–15% and total energy consumption in data centers by 8%–13%. These results demonstrate the effectiveness of the Rock Hyrax-based load balancing algorithm in improving performance and energy efficiency in data centers, highlighting its potential impact on optimizing resource allocation and enhancing overall system performance.

INDEX TERMS Cloud computing, energy consumption, load balancing, makespan, rock hyrax.

I. INTRODUCTION

With the availability of the Internet and the increasing demand for high-performance computing, cloud computing has grown very quickly. The cloud computing provides hardware and software resources to perform their tasks. Cloud computing can be defined as an internet-based model that provides shareable resources such as memory, network, and applications on demand by end users [1]. Cloud service providers dynamically allocate tasks on a virtual machine that

The associate editor coordinating the review of this manuscript and approving it for publication was Mueen Uddin.

is easily accessible to end users. This includes scheduling and allocating available resources for end-user-submitted tasks [2]. The result of this process can be skewed resource utilization, where some resources are overused while others remain unused [3]. If errors such as machine failure are encountered and, additionally, resource usage may not be more appropriate, load balancing among resources becomes critical. Cloud load balancing has an impact on the overall performance of the system. It acts as a compromise between the financial interests of the cloud service provider and the level of end-user satisfaction. Agreements between the cloud service provider and the end user, known as Service Level Agreements (SLAs), are also considered during load balancing.

Most cloud data centers are made to handle a large number of requests. Allocation requests result in high power consumption and low utilization. Resource usage leads to energy consumption, so it must be optimized [5]. Load balancing can be used to balance the load between physical and virtual machines as the tasks are assigned to a cloud data center so that nodes have having equal load. The mapping of tasks to different resources is done to maintain optimized energy consumption [6]. An efficient load-balancing algorithm can enable better resource utilization, helping to reduce the data center's carbon footprint as well as machine cooling requirements. Every physical machine operating in a data center generates heat. When these physical machines have been in use for some time, hot spots are visible in the data centers.

Load balancing can be done in both static and dynamic ways. Static algorithms only work well on nodes where the load doesn't change much. Because of this, these strategies are not efficient in cloud systems that have varied workload requirements. Dynamic algorithms can manage unpredictable workloads and are better than static algorithms [4]. On the other hand, the dynamic algorithm has an additional cost that includes the cost of collecting and storing load information. Dynamic techniques are used to get the right amount of work done on different types of resources.

Load balancing problems can be solved in different ways, which can be grouped into heuristic scheduling, metaheuristic-based scheduling, and hybrid metaheuristics [7]. Each of these methods has its advantages and disadvantages. Metaheuristic scheduling methods may use optimization algorithms like Simulated Annealing (SA) and Ant Colony Optimization (ACO) to make the best schedules. In metaheuristic scheduling, goals are included in the fitness function of the algorithm. Because of this, several objectives are passed to the scheduling process, where each function is distinct. This affects the overall schedule and their respective outcomes. These algorithms were also utilized to explore solutions for a wide range of optimization problems [8]. In the last 20 years, metaheuristic optimization algorithms have become very popular. Some of these algorithms are so well-known that scientists in other fields are familiar with them. They usually get ideas from natural things and how animals act. In addition, these algorithms are easy to understand, use, and put together. The reason to use these algorithms is that they are random and work better than traditional ways of optimizing when compared to local optimizations. One thing that all meta-heuristic approaches have in common is that they divide the problem into two phases: exploration and exploitation [8].

Swarm intelligence (SI) is one of the best-known fields of meta-heuristic algorithms, which have been used in many different ways over the past few decades. The main reason that gave rise to the idea of swarm intelligence was the fact that different types of social swarms behave as a group [9]. The individuals in these swarms work together in a coordinated, interactive system to reach different kinds of goals. The natural behavior of each swarm within the swarms is what makes the interactive system work. SI techniques have been increasingly used recently to solve different kinds of problems in different fields, such as robot control systems and optimization problems for resource management like load balancing problems that require robustness and flexibility. To solve optimization problems, SI techniques such as Particle Swarm Optimization (PSO) [10], Algorithmic Colony Optimization (ACO) [11], and Artificial Bee Colony Optimization (ABC) [12] have been developed. Recent years have seen the development of more complex swarm intelligence optimization algorithms such as the Grey Wolf optimizer [13], the Cuckoo search [14], the Bat algorithm [15], the Firefly algorithm [16] and the Whale optimization algorithm [17], FOX [18], [19], Eagle Strategy [20], [21]. To balance the load on cloud servers, a meta-heuristic optimization method inspired by the behavior of the Rock Hyrax, an African mammal, is proposed in the paper. The proposed algorithm will resolve the following issues:

1) Balance the load on the VM as the tasks and machines vary.
2) Execution of submitted tasks within the deadline while maintaining SLA.
3) A multi-objective load balancing algorithm to balance the load that considers both users and service providers.

Load balancing in the cloud is an NP-complete problem. The task is assigned to the virtual machine using cloud resource scheduling. The purpose of balancing is to efficiently utilize the available resources. However, load balancing is critical for both end users and service providers. As a result, this research takes into account both of their objectives. The paper proposes a meta-heuristic algorithm that helps in achieving objective functions.

The contributions of the paper are the following:

1) A meta-heuristic load-balancing method for cloud environments considering job and resources dynamic and heterogeneous in nature.
2) A multi-objective algorithm considering both users and service providers to minimize energy consumption and makespan.
3) The effectiveness of the proposed algorithm is tested with different tasks and resources.

The rest of the paper is structured as follows: Section II is devoted to the revision of previous relevant works, Section III offers a detailed description of the proposed work, Section IV provides an analysis of the results and, finally, Section VI presents the conclusions of this piece of research.

II. RELATED WORK

The authors in [10] proposed a load balancing algorithm based on Particle Swarm Optimization. This algorithm transfers extra tasks from an overloaded virtual machine. In [28], the authors have used a hybrid algorithm combining cuckoo search with ACO to balance load in a cloud environment. The authors in [29] have used the BAT optimization algorithm for balancing load in the cloud environment.

In [12] the authors have proposed a technique for load balancing that is based on the foraging behavior of honey bees. The algorithm considers the priorities of tasks while balancing the load across VMs. The tasks removed are considered honey bees, and the information is updated globally. The algorithm improves overall throughput by reducing the waiting time for a job. To minimize makespan and execution time, the authors in [15] used the BAT algorithm for balancing load in cloud computing. In [24] the authors gave diary wise cloud calculations for adjusting load issues, including GC (Genetic Calculations), ACO (Ant Colony Optimization), ABC (Artificial Bee Colony) and PSO (Particle Swarm Optimization). They presented another idea, Ant Lion Optimizer (ALO) based for the most part on distributed computing environments, as an affordable standard that was expected to give results in adjusting the heap.

The authors of [30] propose a reinforcement Learning-based parallel PSO for Intelligent Decision-Making. The authors of [26] presented a hybrid algorithm that combines Harries Hawks Optimisation (HHO) with Ant Colony Optimisation (ACO). In [41], the authors provided multi-objective load balancing using fuzzy decision-making and GA.

A load-balancing algorithm in cloud computing was proposed by authors [11] that use the Ant colony optimization technique. They have updated the pheromone mechanism to balance the load and minimize CPU. For detecting hosts that are overutilized, the authors [14] followed an approach based on the Cuckoo Optimization Algorithm and have migrated jobs from such hosts to others. The parameter considered for load balancing is energy consumption.

The authors in [25] proposed a load balancing method based on the hybrid dingo and whale optimisation algorithms (HDWOA-LBM) for optimal throughput and resource utilisation in cloud computing environments. The authors in [26] proposed a method that combines Harries Hawks Optimisation and Ant Colony Optimisation to increase multi-QoS parameter performance. The authors in [27] developed an optimal fuzzy-based load balancing model for effective resource allocation that uses fuzzy variables such as memory, bandwidth, and disc space, implements categorization-based limitations, and assigns jobs to virtual devices.

Heuristics and metaheuristic algorithms are crucial for solving complex problems in fields like logistics, finance, engineering, and AI. The impact of these technologies on optimizing delivery routes and resource allocation is unparalleled. They find quick solutions for large-scale problems, adapt to constraints, and handle uncertainties. They are continuously improved through research and integration with other AI techniques. As the world becomes increasingly complex, heuristics and metaheuristics still remain relevant due to their ability to efficiently tackle the complexities of the ever-evolving world. Heuristic-based stitching is an image processing technique that uses heuristic algorithms and refined rules to create seamless panoramas from multiple images [22].

Load-balancing algorithms in cloud computing are essential for optimizing resource utilization and ensuring high performance. However, they face challenges in handling dynamic workloads, which can lead to suboptimal resource allocations and performance degradation. The heterogeneity of cloud resources, including variations in computing power and network capabilities, also affects load-balancing decisions. The work done in the past considers a single objective function around which the algorithm works. To address these weaknesses, more effective load balancing algorithms are required, including dynamic strategies considering cloud resource heterogeneity and multi-objective functions to support different QoS. We have proposed a Rock Hyrax load balancing algorithm that is dynamic, considers heterogeneous resources and jobs, and supports multiple QoS requirements.

III. ROCK HYRAX ALGORITHM

This section is aimed at proposing an algorithm for load balancing in cloud computing, where the user submits their tasks to the broker. The broker schedules these jobs on the cloud and allocates resources to them. During the runtime, if the load on any server goes beyond the threshold limit, then the proposed algorithm for load balancing is executed. A framework describing the entire process is shown in Figure 1.

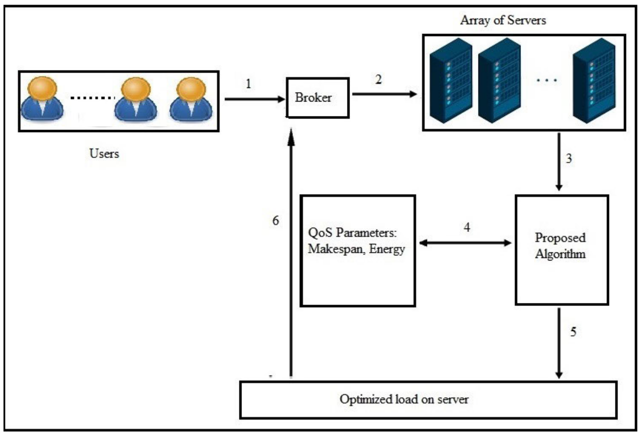


FIGURE 1. Proposed Framework for Load Balancing Algorithm.

The proposed Rock Hyrax algorithm considers QoS parameters such as makespan, response time, throughput and energy efficiency and balances the load on an overloaded server. The workload of the server and its available capacity are fed back up to the broker, who reallocates the waiting tasks on different servers for their execution.

The broker has some m jobs that have to be allocated to n servers. The problem is that before one job executes, another job may come to the server, and the arrival rate of jobs is not fixed. Therefore, it becomes very difficult for the broker to keep track of the number of jobs that have not been allocated to a server. The worst possible scenario is that the arrival rate of jobs can be much faster than the rate of executing the tasks.

One approach can be used to fix a server n_i for similar kinds of jobs m_i. The problem with this approach is that the jobs that are being submitted are different, and each has its own demand for resources for execution. Also, if the number of similar jobs is too high, the server n_i will be overloaded, while other servers $n_n - i$ will be underloaded. Thus, allocating one server for a similar kind of job is no longer feasible. The load balancing algorithm will fail if any server n_i gets more than (m/n) requests, where m is the total number of requests coming to the broker. If the jobs that are coming are similar, then, as per the approach, the majority of jobs will be allocated to a single resource, increasing its chance for failure. Therefore, the proposed Rock Hyrax Algorithm calculates an upper bound on the expected load of a server, n_i such that the server is not overloaded. The proposal uses the hash function h to achieve optimal load on the server.

The proposed algorithm works in two phases: estimation of server capacity and load balancing by the Rock Hyrax algorithm. The detailed computation and algorithm are presented in the next subsections:

A. ESTIMATION OF SERVER CAPACITY

For allocating a job to a server from m number of jobs, a job m_i is selected in such a way that m_i hashes to 1 for the chosen hash function h. A modulo operator is applied to the hash function where the modulus value is equal to the resources available. The upper bound on other jobs is also calculated so that all jobs that hash to 1 are found. All those jobs that will hash to 1 will be allocated to a server, while others will wait in the waiting queue. The performance of the algorithm depends on the value of the threshold. If the value of the threshold is too small, the load balancing algorithm will be called too frequent, and if it is too high, then the makespan will become high. Therefore, it is important to choose an optimal value for the threshold.

B. CALCULATION OF HASH FUNCTION AND JOB ASSIGNMENT

To calculate the threshold value of the server for defining its load capacity, a random variable t is chosen. For the calculation of random variables, several equations were used. The broker can do some basic calculations of mapping the job m_i to a server n_i by any mathematical function f(i), where f is a mapping function. Equation 1 selects all the jobs that hash to 1 so that the broker can allocate the resources to them. The selection of the random variable t is crucial in determining the threshold value for the server's load capacity. By utilizing various equations, the calculation of random variables becomes possible. These equations aid in accurately mapping each job m_i to its corresponding server n_i through the mathematical function f(i).

$$f_{h(y)=1} = \begin{cases} 0, & \text{if} \;\; h(y) \neq 1. \\ 1, & \text{if} \;\; h(y) = 1. \end{cases} \tag{1}$$

The mapping function only considers the job for the allocation of resources if the value of the hash function is 1; otherwise, it places the job in the waiting queue. The same is illustrated in Equation 1. All those jobs that hash to a value of 1 calculated in Equation 1 are collected in H. H is calculated as

$$H = \sum_{y \in U} f_{h(y)=1} \tag{2}$$

where U is the universal set and U $= 1, 2, 3 \ldots . m$.

The set of all jobs that satisfy the condition of having a hash value of 1. This set is then used for further processing and allocation of resources, while the remaining jobs are kept in the waiting queue until their hash value becomes 1. H is the subset of submitted jobs such that all jobs that hash to one are included in H. When the hash function is calculated for the jobs, all jobs that have a hash value of 1 are assigned to a server. Such jobs are kept in the subset given by H.

C. JOB LIMIT PER SERVER

As jobs m varies with servers n are fixed, and the mapping of jobs to a server is one to many, we need to find the load on a server that can it take. For estimating the expected load on a server, a variable $\mathbb{E}_h$ is defined. The value of $\mathbb{E}_h$ will be different for each server depending upon various factors, such as processor speed. To estimate the load on server h, the number of jobs assigned to it by the broker, the value of the expected load on servers $\mathbb{E}_h$ can be defined as:

$$\mathbb{E}_h[H] = \mathbb{E}[\sum_{y \in U} f_{h(y)=1}] \tag{3}$$

D. PROBABILITY OF ASSIGNING A JOB

The expected value is the probability of multiple occurrences of an event. So, the estimated load on server h can be calculated by converting $\mathbb{E}$ into a probability function, where the probability function defines the probability of a job m_i allocated to a server h. Thus, by converting $\mathbb{E}$ into a probability function, Equation 4 mathematically is expressed as:

$$\mathbb{E}_h[H] = \mathbb{E}[\sum_{y} Pr[h(y) = 1]] \tag{4}$$

Since the jobs are considered to be heterogeneous, for calculation, we define two sets of jobs x and y such that $y \neq x$. Thus, the probability function calculated in Equation 4 is a combination of two probabilities for job x and job y. The estimated probability of the hash function that hashes to 1 $Pr[h(y) = 1]$ for all jobs $y \neq x$ can be given by adding the probability of job x and job y. Mathematically, this is expressed as:

$$\mathbb{E}_h[H] = Pr(h(x) = 1) + \mathbb{E}[\sum_{y \neq x} Pr[h(y) = 1]] \tag{5}$$

This allows to account for the different characteristics and requirements of each job in the calculation of the probability function. By considering both job x and job y, we can obtain a more comprehensive estimate of the probability of the hash function resulting in 1 for all jobs except x.

Expressing the estimated probability for the hash function $\mathbb{E}[H]$, in terms of total requests received by the broker and total requests handled by a server h, can be further expressed as:

$$\mathbb{E}_h[H] \leq 1 + \frac{tr - 1}{m} \tag{6}$$

where, tr is the total requests handled by a particular server 1, and m is the total number of requests received by the cloud broker.

To estimate the deviation of load t on servers, it can be expressed by the difference of total jobs that hashes to 1 from H and the estimated probability of a job being allocated to server $\mathbb{E}$[H]. For any server h, this can be mathematically written as:

$$H - \mathbb{E}_h[H] > t \tag{7}$$

The goal of the load balancing algorithm is to minimize this value of t because a higher value of t indicates a load imbalance on the server. This can be calculated as a probability of deviation on load. Thus, we have to calculate the variance of the load on the server. Using Chebyshev's inequality statement [31] which states that for any probability distribution, there can be only a few certain values that will be far from the mean, we applied the concept of inequality statement, for any random variable t

$$Pr[|H - \mathbb{E}[H]| > t\,\mathbb{E}[H]] \leq \frac{Var[H]}{x^2\,\mathbb{E}^2[H]} \tag{8}$$

where $Var[H]$ is the variance of the hash function H. By definition, variance can be defined as:

$$Var[H] = \mathbb{E}[H^2] - \mathbb{E}[H]^2 \tag{9}$$

E. VARIANCE OF THE WORKLOAD

To find the $Var[H]$, we need to calculate the values of $\mathbb{E}[H]^2$ and $\mathbb{E}[H^2]$. Once the value of $\mathbb{E}[H]$ has been calculated, its square will give $\mathbb{E}[H]^2$. Therefore, we need to calculate the value of $\mathbb{E}[H^2]$. $\mathbb{E}[H^2]$ can be expressed as the summation of the hash functions of jobs x and y. Mathematically, $\mathbb{E}[H^2]$ can be expressed as:

$$\mathbb{E}[H^2] = \mathbb{E}[\sum_x f_{h(x)=1} \sum_y f_{h(y)=1}] \tag{10}$$

F. EXPECTATION THAT TWO DIFFERENT JOBS WILL BE ASSIGNED TO THE SAME SERVER

Expressing the values of $\sum_x f_{h(x)=1}$, as probability function. For all jobs $x \neq y$, the probability of jobs hashing to 1 can be expressed as:

$$\sum_x f_{h(x)=1} = Pr([h(y) = 1] + \sum_{x \neq y} Pr[h(x) = 1]) \tag{11}$$

Similarly for jobs $y \neq x$, the value of $\sum_y f_{h(y)} = 1$ can be expressed as:

$$\sum_y f_{h(y)=1} = Pr([h(x) = 1] + \sum_{y \neq x} Pr[h(y) = 1]) \tag{12}$$

Calculating the value of $\sum_y f_{h(y)=1}$ and $\sum_x f_{h(x)=1}$ by the estimated probability for the hash function $\mathbb{E}$[H] for jobs x having a hash value of 1, $\sum_x f_{h(x)=1}$ is calculated as:

$$\sum_x f_{h(x)=1} \leq 1 - \frac{m-1}{n} \tag{13}$$

and $\sum_y f_{h(y)=1}$ for jobs y having a hash value of 1 is calculated as:

$$\sum_y f_{h(y)=1} \leq 1 + \frac{m-1}{n} \tag{14}$$

Using these calculated values of $\sum_y f_{h(y)=1}$ and $\sum_x f_{h(x)=1}$ the value of $\mathbb{E}[H^2]$ is calculated as:

$$\mathbb{E}[H^2] = 1 - (\frac{m-1}{n})^2 \tag{15}$$

G. CALCULATION OF THE PROBABILITY THAT A JOB WILL BE ASSIGNED TO A SERVER

Since the variance of a random variable gives the possible values of that variable. Therefore, $Var[H]$ can be calculated by using the calculated values of $\mathbb{E}[H^2]$ and $[\mathbb{E}[H]]^2$. The $Var[H]$ is:

$$Var[H] = \frac{m-1}{n}(1 - \frac{m-1}{n}) \tag{16}$$

This value of $Var[H]$ is the threshold value for any server to accept jobs. If the value exceeds the limit value, the server will be declared an overloaded server.

A pseudocode describing the process of estimation of server capacity is illustrated in Algorithm 1.

The first phase of the algorithm is to find virtual machines that have loads greater than their respective threshold values. The proposed algorithm checks every virtual machine initially, and as soon as a job is transferred to that virtual machine, it rechecks the virtual machine. The threshold is defined by the queue length of waiting jobs. Listing 1 uses nodes' real-time workload and resource states to make informed decisions on load distribution. It involves dynamic load monitoring, triggering load-balancing decisions based on predefined thresholds. When a new job arrives, the queue length is checked, and if it exceeds the threshold, the algorithm is called; otherwise, the job is executed by the VM. Upon identifying the capacity of server$_j$ with the status of overload

Algorithm 1 Process of Proposed Algorithm Phase I

```
Input: Jobs $m$, Server $n$, Threshold_Value
Initialization();
i,j = 0
For $i = 1$ to $m${
   For $j = 1$ to $n${
      For $Job\{m_i\}$ allocated to $server\{n_j\}${
        JobQueueLength $+= 1$
        If JobQueueLength $>$ Threshold_Value{
           Call RockHyrax Load Balancing Algorithm
        }
      }
   }
}
Return the servers $n_i$
Result: Servers $n_i$ having high load
```

and underload, it is imperative to balance the load on the identified overloaded server. Therefore, the next subsection discusses the proposed Rock Hyrax Algorithm for load balancing in cloud computing.

H. ROCK HYRAX LOAD BALANCING ALGORITHM

This section introduces a load balancing mechanism based on Rock Hyrax [32] behaviour. Rock Hyrax is a swarm optimization-based metaheuristic algorithm [42]. The Rock Hyraxes are tiny, vegetatrian mammals that dwell in the Middle East and Africa [33]. They frequently forage in flocks in the mid-morning or evening. Rock Hyrax groups often have 80–100 members. During feeding, one party member glances around to defend other members from predators [34].

Rock Hyraxes foraging behavior mimics Split and Conquer, with the male responsible for the source and informing the group after successful feeding. To make the best decisions for the group, a limit is maintained on feeding behavior [35]. Rock Hyraxes communicate with each other using different sounds, each with a different meaning. One sound can signal an alarm, while the other signals food. To mark the territory and the path of food, Hyraxes use the odor secreted by the dorsal gland [35].

The process flow for the proposed Rock Hyrax load balancing algorithm is shown in Figure 2.

In cloud computing, end users submit their jobs to brokers. The broker allocated the resources to jobs for execution as per the SLA [23]. As the job moves to the resource (VM) for execution, the length of the waiting queue in a virtual machine is increased by one. Once the length is increased by one, the algorithm for checking whether the server has exceeded its threshold value is checked. If queue length exceeds the threshold value, the VM is declared overloaded, and in phase II of the proposed algorithm, the Rock Hyrax algorithm is executed. The algorithm initializes the population in the specified problem space. Subgroups are formed from this entire population, and each sub-group can have 80–100

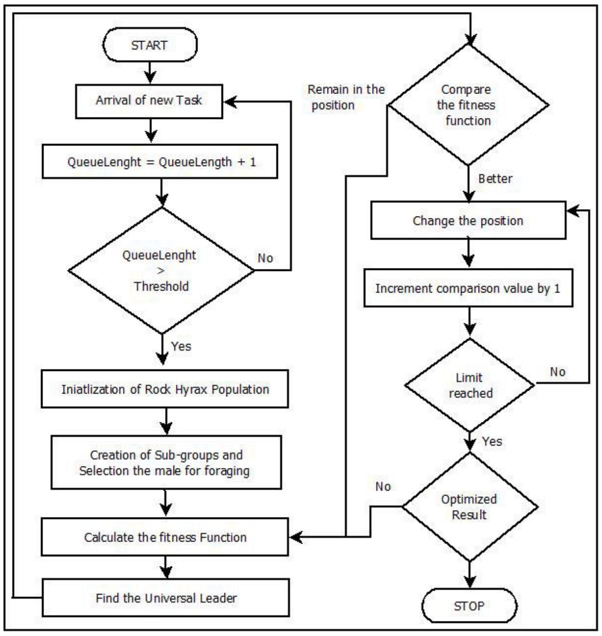

START
Arrival of new Task
QueueLenght = QueueLength + 1
QueueLenght > Threshold
No
Yes
Iniatlization of Rock Hyrax Population
Creation of Sub-groups and Selection the male for foraging
Calculate the fitness Function
Find the Universal Leader
Remain in the position
Compare the fitness function
Better
Change the position
Increment comparison value by 1
Limit reached
No
Yes
Optimized Result
No
STOP

FIGURE 2. Process flow of the Proposed Algorithm.

Rock Hyraxes. The algorithm calculates the fitness function of every Hyrax in the group, and based on the value it selects the universal leader or privileged Rock Hyrax. The universal leader or privileged Rock Hyrax, is responsible for foraging food or searching VMs that are underloaded. The subsequent iterations are called and the fitness function is recalculated. If the value of the fitness function is better, the privileged Rock Hyrax changes its position and increments the limit for iteration by one. If the limit is reached, it checks for the fitness function. If the fitness function is optimized, the algorithm is stopped; otherwise, the fitness function is recalculated. The pseudocode of the algorithm is given in Listing 2.

The data structure used in the proposal is as follows:

- Neighbour$_{init}$ defines the size of the neighbourhood, which will decrease after each iteration completion with a fixed value.
- RH$_{num}$ is the entire population of Rock Hyrax, VM$_{num}$ is the available number of VMs in the environment.
- PrivilegedVM$_{num}$ is the number of overloaded VMs and PrivilegedVM$_{num}$ < VM$_{num}$.
- PrivilegedRH$_{num}$ is a common lead found in iteration and is smaller than RH$_{num}$.
- OtherVM$_{num}$ is the difference between VM$_{num}$ and PrivilegedVM$_{num}$.
- γ is a decomposition function that controls the rate of odor evaporation.

The Rock Hyrax set RH$_i$ is initialized using a symmetric distribution in the problem space, as the Rock Hyrax is present in the same position. The population of Rock Hyrax is obtained by multiplying the problem space by a symmetrical distribution.

$$RH_i = RH_i - S(0,1) * Problem_{size} \tag{17}$$

For Rock Hyraxes to decide which path to choose, a probability P_i is defined. P_i is calculated by dividing the fitness value of the universal Rock Hyrax by the cost of the chosen path. To normalize the value, a multiplication factor of 0.9 is used. Mathematically, the probability P_i is calculated as:

$$P_i = 0.9 * RH_{best}/Cost_{sh} \tag{18}$$

The Rock Hyrax algorithm efficiently manages server load balancing by intelligently assigning submissions to available resources and constantly monitoring for optimal load balancing. The rock hyrax represents the cloudlets or jobs, while food represents resources. In the event of an overload, the algorithm activates the Rock Hyrax population and selects the job with the lowest cost to be transferred to an available resource. The algorithm then relentlessly searches for the best available resource until the fitness function is fully optimized. With each successful job schedule, the population size is reduced by a factor. The various parameters used in Algorithm 2 are described in Table 1.

TABLE 1. Data Structure used in Algorithm.

Parameters	Description
Nsize	Depicts problem space and is reduced by a constant value after each iteration
RHtotal	Population of Rock Hyrax in problem space
VMtotal	Number of VMs available
SelectedVMtotal	VM selected to schedule a job and ∈ VMtotal
UniversalRHtotal	Rock Hyrax having best fitness value and ∈ RHtotal.
OtherVMnum	The difference between VMtotal and SelectedVMtotal
Ch	The cost of scheduling jobs
RHbestcost	The final cost of job scheduling
VMbest	Best VM to map a job found during iteration
VMsize	The numbers of VMs used to create a neighbourhood after a successful mapping
NeighbourDecreasefactor	Factor that is used to create neighbour size after every unsuccessful mapping
n	Limit for iteration

Algorithm 2 Proposed Rock Hyrax Load Balancing Algorithm

```
Data: $Problem_{size}$, $RH_{num}$, $VM_{num}$, $PrivilegedVM_{num}$, $Neighbour_{init}$, $PrivilegedRH_{num}$, $OtherVM_{num}$
Initialization();
Population ← InitializePopulation(RH$_{num}$, Problem$_{size}$)
RHbest$_{cost}$ ← Cost(S$_h$)
Odor$_{init}$ ← 1.0 /(Problem$_{size}$ * RHbest$_{cost}$)
Odor ← InitializeOdor(Odor$_{init}$)
While (StopConditon()){
  EvaluatePopulation(Population);
  RH$_{best}$ ← GetBestSolution(Population);
  NextGeneration ← $\phi$;
  Neighbour$_{size}$ ← (NeighbourSize$_{init}$
     * NeighbourDecrease$_{factor}$);
  VM$_{best}$ ← SelectBestVM(Population, VM$_{num}$);
  ForEach $VM_i \in M_{best}${
     SelectedRH$_{num}$ ← $\phi$;
     If $i < PrivilegedVM_{num}${
        $SelectedRH_{num}$ ← $PrivilegedRH_{num}$;
     }
     Else{
        $SelectedRH_{num}$ ← $OtherRH_{num}$;
     }
     $Neighbourhood$ ← $\phi$;
     For $j$ to $RecuritedRH_{num}${
       $Neighbourhood$ ←
          CreateNeighbourhoodRH($Site_i$,$VM_{size}$);
     }
     NextGeneration←
     GetBestSolution(Neighbourhood);
  }
  $RemainingRH_{num}$ ← (RH$_{num}$ - VM$_{num}$);
  For ($j$ to $RemainingRH_{num}$){
     NextGeneration ← CreateRandomRH();
  }
  Population ← NextGeneration;
  GlobalUpdateOdor(Odor, RH$_{best}$, RHbest$_{cost}$, $\gamma$);
}
Return $RH_{best}$
Result: $RH_{best}$
```

IV. RESULT ANALYSIS

The proposed load balancing algorithm is inspired by nature-inspired algorithms and uses parameters like finding the overloaded server, selecting the server for the transfer of jobs, and calculating the fitness function to balance the load on the cloud.

A. EXPERIMENTAL SETUP

For simulating the environment, CloudSim version 3.0.3 on Windows 7 was used. CloudSim is a library that simulates a cloud computing environment. CloudSim was used to create VMs, data centers, and cloudlets to evaluate the load balancing algorithm. In the simulation, four data centers were created to emulate the environment of the proposed algorithm.

B. PERFORMANCE EVALUATION

Various QoS parameters are used to measure the performance of a service in the cloud. The QoS parameters that are considered in the paper for measuring the performance of the proposed algorithm are described in this section.

1) MAKESPAN

Makespan is the maximum execution time for all tasks submitted to the cloud for execution. The value of makepan must be minimal so that all tasks are executed within the request timeout [36]. This can be expressed as:

$$MS_{TIME} = T_i(CT_{TIME}) \tag{19}$$

where
MS_{TIME} is the makespan time of $task_i$
CT_{TIME} is the completion time of $task_i$

2) THROUGHPUT

Throughput is the rate of execution of tasks in cloud computing in a specified period of time [37] and can be represented as:

$$Throughput = \sum_{i=1}^{n} ET_i \tag{20}$$

where ET_i is the execution time of task i

3) RESPONSE TIME

Response time is the total time between the send and wait times of jobs in the queue, indicating the cloudlet response required for a specified entry [38] and can be expressed as follows:

$$RT_{TIME} = T_i(ST_{TIME} + WT_{TIME}) \tag{21}$$

where,
RT_{TIME} - the response time for task i
ST_{TIME} - the submission time for task i
WT_{TIME} - the waiting time for task i.

4) ENERGY CONSUMPTION

Energy is the entire amount of energy required by a server to schedule a task in the cloud. The algorithm performs better when its power consumption is modest. You must first determine the power consumption of the actual machine in order to calculate the power consumption. [39], [40], the power model is suggested as:

$$PW(u) = PW_{idle} * PW_{max} + (1 - PW_{idle}) * PW_{max} * u \tag{22}$$

where
PW is the power utilized for a time t when the CPU is being utilized (u).
PW$_{max}$ is the maximum power consumed by a machine under 100% utilization.
PW$_{idle}$ is the amount of power consumed by the machine under idle state.
Thus, the energy consumed by a machine is represented by

$$E = \int_t PW(u(t)) \tag{23}$$

where
E is Energy Consumption
As the problem is formalized as a multiobjective, this paper considers makespan and energy as our fitness functions while balancing the load. Thus, the fitness function can be formalized as a function of minimization and can be given as:

1) Minimization of makespan time.

$$MS_{TIME} = Min(T_i(CT_{TIME})) \tag{24}$$

2) Minimization of Energy Consumption

$$E = Min(\int_t PW(u(t))) \tag{25}$$

C. EXPERIMENT RESULT

In this section, we will discuss the outcomes of applying the Rock Hyrax algorithm that was proposed to two different scenarios. In scenario I, the number of virtual machines remains constant, but the number of jobs increases by 10 with an increment of 10. Table 2 contains a CloudSim experimental configuration for performing the algorithm in Scenario I. Virtual machines employ time-sharing to share resources in four data centers. Also, in Scenario II, the number of tasks is fixed, the number of VMs changes from 10 to 100 in 10 increments, and the system uses a time-sharing policy to share resources for tasks. Each virtual machine has 1 GB of RAM.

TABLE 2. Experimental Setup for Both Scenarios.

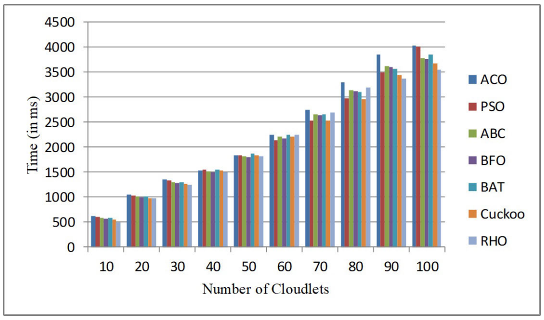
Entity Types	Variable	Scenario I	Scenario II
User	Cloudlets	10-100	500-10000
Cloudlets	Length	500-10000	250-10000
Host	Hosts	4	4
	RAM	4GB	4GB
	Storage	40GB	40GB
	Bandwidth	512	512
VM	VMs	8	512
	RAM	2 GB	2 GB
	OS	Windows	Windows
	Policy	Time sharing	Time sharing
	CPUs	4	4
Data Centers	Data Centers	2	2

Since QoS describes the performance of services for load balancing in cloud computing. Therefore, for result analysis purposes, we have considered makespan time, energy efficiency, throughput, and response time as QoS parameters that will be judged upon various approaches. Further, for validation purpose Rock Hyrax Algorithm is compared with various prevalent approaches like Ant Colony Optimization [11], Particle Swarm Optimization [10], Artificial Bee Colony Optimization [12], Bat Algorithm [15] and Cuckoo Algorithm [14].

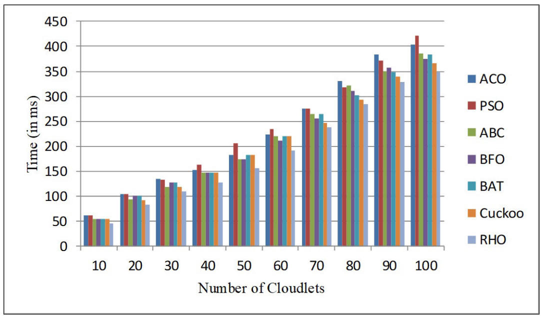
D. RESULT FOR SCENARIO I

Figures 3, 4 depict the proposed algorithm's QoS-based performance comparison for the scenario I, where virtual machines are constant and the variable value of tasks is 10 and 50 respectively. The suggested technique performs similarly to the comparison algorithm when there are few cloudlets. It maintains a constant makespan time as the number of cloudlets increases, but algorithms like ACO and PSO become less effective as the number of cloudlets increases. The method selects the optimal fitness function without resulting in a shorter makespan.

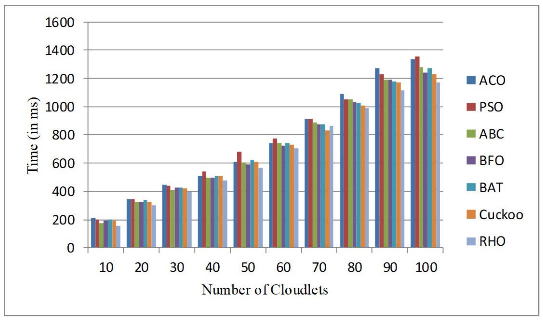
Figures 3(a) and 4(a) shows increase in makespan time as cloudlets increases. The proposed strategy attempts to strike a balance between utilizing existing resources and seeking out



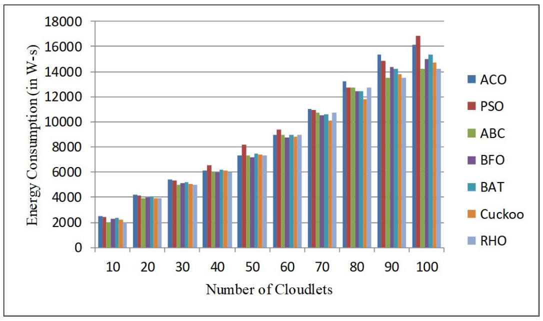
(a) Makespan Time



(b) Throughput

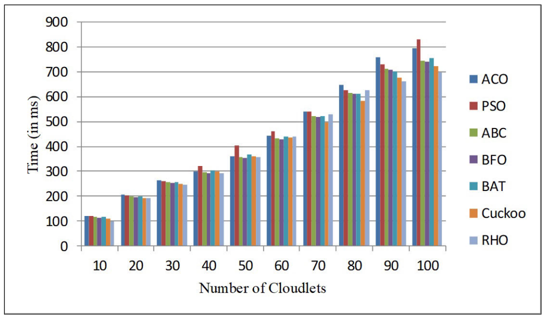


(c) Response Time

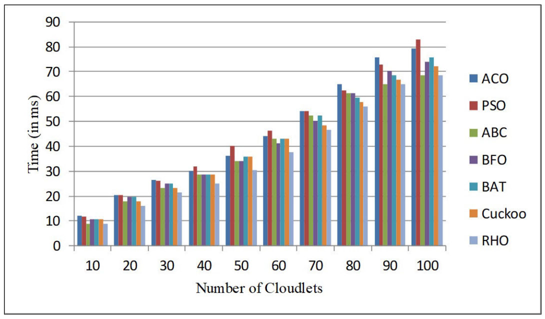


(d) Energy Consumption

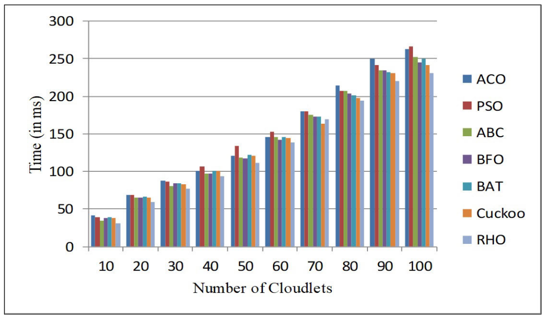
FIGURE 3. Performance evaluation for different QoS with varied jobs and VM = 10.



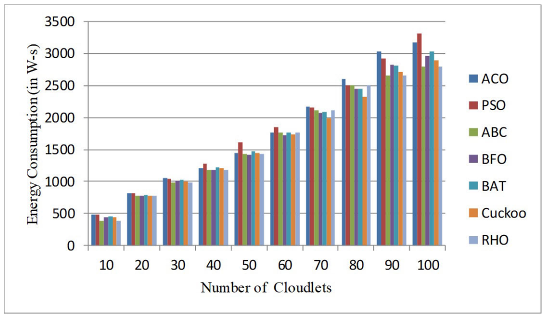
(a) Makespan Time



(b) Throughput



(c) Response Time



(d) Energy Consumption

FIGURE 4. Performance evaluation for different QoS with varied jobs and VM = 50.

new ones. This strategy, unlike earlier ones, offers a higher convergence rate while locating the optimal solution and producing overall superior outcomes. Figures 3(b) and 4(b) show that the proposed method has a higher throughput than the other algorithms when the number of virtual machines (VMs) that can be used to schedule work is changed. Throughput can go up with the proposed solution because jobs in the queue will have to wait less. The proposed algorithm optimizes virtual machine idle time by efficiently allocating tasks, and enhancing system performance. This leads to increased productivity and reduced costs for organizations relying on task scheduling and resource allocation. The algorithm considers the resource utilization of each virtual machine, preventing overloading and enhancing overall performance. This improves throughput, user satisfaction, and response times for critical tasks, thereby reducing bottlenecks and enhancing overall system performance.

Figures 3(c) and 4(c) show that the suggested method gives more accurate estimates of reaction time than methods that have been used in the past. This is because the technique reduces the amount of time jobs must wait in the server's waiting queue by informing the broker of the server's available and current space. By providing real-time information about the server's availability and capacity, the suggested method enables the broker to make more informed decisions regarding job allocation. This optimization in job scheduling ultimately leads to reduced waiting times and, consequently, more accurate estimates of reaction time.

Figures 3(d) and 4(d) show how well algorithms use energy when adjusting the number of virtual machines (VMs) for a fixed number of workloads. The proposed method can cut down on energy use, increase throughput, and cut down on the average amount of time it takes to finish a task. Also, the server arranges jobs by observing their busiest times. As a result, the number of idle servers has decreased. Therefore, less energy is consumed when the system is idle. This improves the overall energy efficiency of the system.

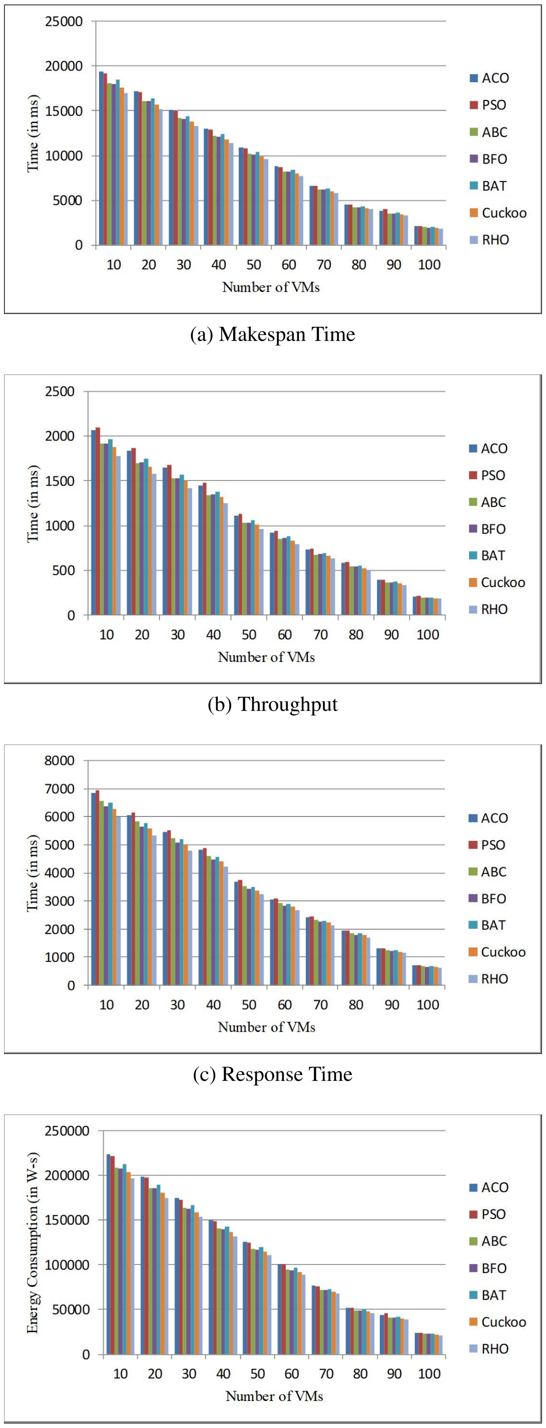
Additionally, the proposed method dynamically scales the number of VMs based on workload demands, ensuring optimal resource allocation and minimizing wastage. This adaptive approach allows for efficient utilization of server resources, reducing energy consumption during peak and off-peak periods. Consequently, the system achieves a higher level of energy efficiency and cost savings.

Furthermore, the dynamic scaling of VMs also improves the system's performance by ensuring that there are enough resources available to handle high workload demands. This prevents any potential slowdowns or bottlenecks in the system, resulting in a smoother and more efficient operation overall. Additionally, by reducing energy consumption and optimizing resource allocation, the proposed method contributes to a more sustainable and environmentally friendly computing environment.

E. RESULT FOR SCENARIO II

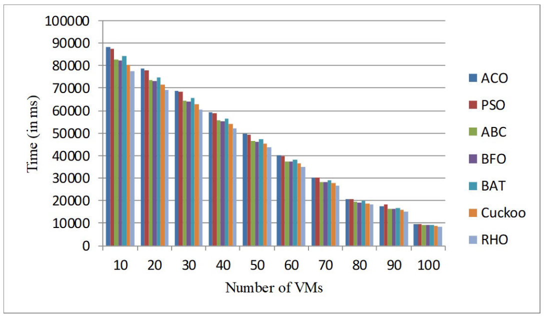
The proposed approach for scenario II depicted in Figures 5 and 6, where the number of cloudlets is constant and the number of virtual machines (VMs) is varied to 500 and 2500, shows a decrease in QoS parameters as the number of VMs increases from 10 to 100. This is due to algorithms finding underloaded VMs for job execution. However, algorithm performance becomes crucial as there are fewer VMs available. The makespan time decreases as VMs number increases. The algorithm assigns jobs evenly to resources with the fewest jobs, allowing all VMs to function together. This results in the algorithm outperforming traditional algorithms in terms of execution time. Additionally, the suggested algorithm also ensures that the workload is distributed evenly among all virtual machines, maximizing their efficiency and reducing the overall execution time. This makes it a more reliable and efficient choice compared to algorithms that may prioritize certain resources over others, leading to longer execution times.

The value of throughput tends to increase as the amount of execution time a cloud computing service has been in operation decreases. This is because as a cloud computing service operates for a longer duration, it becomes more

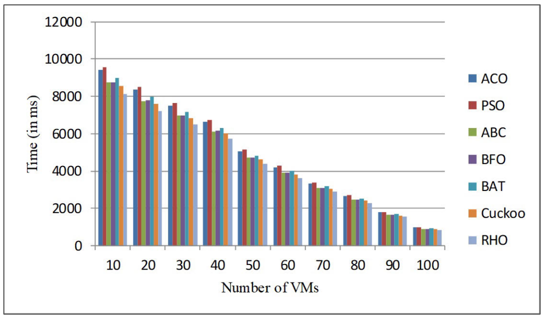


(a) Makespan Time
(b) Throughput
(c) Response Time
(d) Energy Consumption

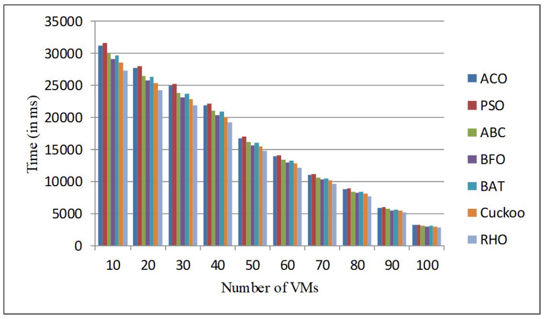
FIGURE 5. Performance evaluation for different QoS with varied VMs and jobs = 500.



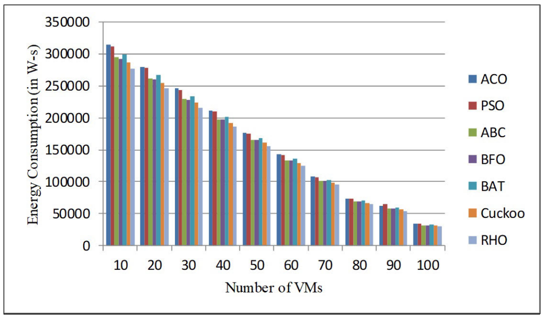
(a) Makespan Time



(b) Throughput



(c) Response Time



(d) Energy Consumption

FIGURE 6. Performance evaluation for different QoS with varied VMs and jobs = 2500.

efficient in managing resources and optimizing its operations. Additionally, the experience gained over time allows for better allocation of resources and improved performance, resulting in higher throughput. The response time can be reduced by identifying underutilized resources using the proposed method. By identifying underutilized resources, the cloud computing service can allocate them more effectively, leading to faster response times. This method helps in identifying any bottlenecks or inefficiencies in resource allocation and allows for their optimization, ultimately reducing response time. If there are any idle servers, the system will assign work to them so that there are fewer servers and less downtime.

As the time required to execute jobs decreases, there is a reduction in the amount of energy that is consumed. As the server execution time is optimized, the suggested algorithm becomes more energy efficient as the number of jobs increases. Additionally, by optimizing server execution time and reducing response time, the suggested algorithm can also improve overall system scalability. This means that as the number of jobs increases, the system can handle a larger workload without compromising performance or energy efficiency. Ultimately, this leads to a more sustainable and cost-effective solution for resource allocation in service-based environments.

V. LIMITATIONS AND FUTURE WORK

Load balancing algorithms in cloud computing face limitations, such as adapting to dynamic workloads and the heterogeneity of cloud resources. The performance of these algorithms remains consistent when the number of jobs or VMs is low, but significantly changes when the variation is too large. The heterogeneity of cloud resources, encompassing diverse computing capabilities, also presents challenges for load-balancing algorithms, as they may struggle to effectively utilize these varying capacities. Also, considering multi-QoS parameters affects the way the experiments are conducted and performed. A trade-off needs to be found so that one parameter does not affect the other.

Network latency is often overlooked in load distribution experiments in complex cloud environments, causing security concerns due to sensitive information exchange. Privacy-preserving strategies are needed to address these issues. The dynamic nature of cloud environments adds complexity to load-balancing algorithms, necessitating continuous monitoring and adjustment. The increasing scale and complexity of cloud networks necessitate innovative load-balancing techniques to handle modern applications' growing demands.

Future research in load balancing algorithms aims to improve workload prediction accuracy and dynamically adapt to changing conditions in cloud computing systems. Machine learning techniques will enhance workload prediction accuracy, while heterogeneous resource-aware algorithms will optimize task assignments. Energy-efficient load balancing strategies and privacy-preserving mechanisms will be crucial for evolving load balancing algorithms to meet the growing demands of dynamic and secure cloud computing environments.

VI. CONCLUSION

This paper proposes a Rock Hyrax optimization method for load balancing in cloud computing. The method distributes jobs among multiple virtual machines based on availability and current load, reducing total makespan time and improving energy efficiency. The algorithm efficiently manages resources, even in geographically scattered data centers. It solves the problem of local maxima, impacting load balancing algorithms' performance. The Rock Hyrax algorithm is proven to work well for both jobs and virtual machines, both statically and dynamically. Compared to other load balancing techniques, the algorithm reduces makespan by 10%-15% and total energy consumption by 8%-13%. This suggests the Rock Hyrax algorithm's effectiveness in improving job and virtual machine performance, enhancing overall efficiency in data centers.

REFERENCES

[1] R. B. Bohn, "NIST cloud computing reference architecture," in *Proc. IEEE World Congr. Services*, Jul. 2011, pp. 594–596.

[2] J.-T. Tsai, J.-C. Fang, and J.-H. Chou, "Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm," *Comput. Oper. Res.*, vol. 40, no. 12, pp. 3045–3055, Dec. 2013.

[3] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1107–1117, Jun. 2013.

[4] P. Beniwal and A. Garg, "A comparative study of static and dynamic load balancing algorithms," *Int. J. Advance Res. Comput. Sci. Manage. Stud.*, vol. 2, no. 12, pp. 1–7, 2014.

[5] A. Uchechukwu, K. Li, and Y. Shen, "Energy consumption in cloud computing data centers," *Int. J. Cloud Comput. Services Sci.*, vol. 3, no. 3, pp. 31–48, 2014.

[6] L. Luo, W. Wu, D. Di, F. Zhang, Y. Yan, and Y. Mao, "A resource scheduling algorithm of cloud computing based on energy efficient optimization methods," in *Proc. Int. Green Comput. Conf. (IGCC)*, Jun. 2012, pp. 1–6.

[7] M. Masdari, S. ValiKardan, Z. Shahi, and S. I. Azar, "Towards workflow scheduling in cloud computing: A comprehensive analysis," *J. Netw. Comput. Appl.*, vol. 66, pp. 64–82, May 2016.

[8] A. Mozaffari, M. Emami, and A. Fathi, "A comprehensive investigation into the performance, robustness, scalability, and convergence of chaos-enhanced evolutionary algorithms with boundary constraints," *Artif. Intell. Rev.*, vol. 52, no. 4, pp. 2319–2380, 2019.

[9] F. Wahid and R. Ghazali, "Hybrid of firefly algorithm and pattern search for solving optimization problems," *Evol. Intell.*, vol. 12, no. 1, pp. 1–10, Mar. 2019.

[10] F. Ramezani, J. Lu, and F. K. Hussain, "Task-based system load balancing in cloud computing using particle swarm optimization," *Int. J. Parallel Program.*, vol. 42, no. 5, pp. 739–754, Oct. 2014.

[11] R. Mishra, "Ant colony optimization: A solution of load balancing in cloud," *Int. J. Web Semantic Technol.*, vol. 3, no. 2, pp. 33–50, Apr. 2012.

[12] L. D. D. Babu and P. V. Krishna, "Honey bee behavior inspired load balancing of tasks in cloud computing environments," *Appl. Soft Comput.*, vol. 13, no. 5, pp. 2292–2303, May 2013.

[13] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Adv. Eng. Softw.*, vol. 69, pp. 46–61, Mar. 2014.

[14] M. Yakhchi, S. M. Ghafari, S. Yakhchi, M. Fazeli, and A. Patooghi, "Proposing a load balancing method based on cuckoo optimization algorithm for energy management in cloud computing infrastructures," in *Proc. 6th Int. Conf. Modeling, Simulation, Appl. Optim. (ICMSAO)*, May 2015, pp. 1–5.

[15] B. Raj, P. Ranjan, N. Rizvi, P. Pranav, and S. Paul, "Improvised bat algorithm for load balancing-based task scheduling," in *Advances in Intelligent Systems and Computing*. Cham, Switzerland: Springer, 2018, pp. 521–530.

[16] X. S. Yang and A. Slowik, "Firefly algorithm," in *Swarm Intelligence Algorithms*. Boca Raton, FL, USA: CRC Press, 2020, pp. 163–174.

[17] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Adv. Eng. Softw.*, vol. 95, pp. 51–67, May 2016.

[18] H. Mohammed and T. Rashid, "FOX: A FOX-inspired optimization algorithm," *Appl. Intell.*, vol. 53, no. 1, pp. 1030–1050, Jan. 2023.

[19] B. M. H. Zade and N. Mansouri, "Improved red fox optimizer with fuzzy theory and game theory for task scheduling in cloud environment," *J. Comput. Sci.*, vol. 63, Sep. 2022, Art. no. 101805.

[20] S. K. Gavvala, C. Jatoth, G. R. Gangadharan, and R. Buyya, "QoS-aware cloud service composition using eagle strategy," *Future Gener. Comput. Syst.*, vol. 90, pp. 273–290, Jan. 2019.

[21] S. S. Mohapatra, R. R. Kumar, and J. Pradhan, "Hybrid eagle strategy for QOS-based cloud service composition," *J. Inf. Optim. Sci.*, vol. 43, no. 5, pp. 1047–1059, Jul. 2022.

[22] K. Prokop and D. Połap, "Heuristic-based image stitching algorithm with automation of parameters for smart solutions," *Expert Syst. Appl.*, vol. 241, May 2024, Art. no. 122792.

[23] S. Singhal and A. Sharma, "Load balancing algorithm in cloud computing using mutation based PSO algorithm," in *Proc. Int. Conf. Adv. Comput. Data Sci.*, 2020, pp. 224–233.

[24] A. A. Salah Farrag, S. A. Mahmoud, and E. S. M. El-Horbaty, "Intelligent cloud algorithms for load balancing problems: A survey," in *Proc. IEEE 7th Int. Conf. Intell. Comput. Inf. Syst. (ICICIS)*, Dec. 2015, pp. 210–216.

[25] K. Ramya and S. Ayothi, "Hybrid dingo and whale optimization algorithm-based optimal load balancing for cloud computing environment," *Trans. Emerg. Telecommun. Technol.*, vol. 34, no. 5, p. e476, May 2023.

[26] M. Sumathi, N. Vijayaraj, S. P. Raja, and M. Rajkamal, "HHO-ACO hybridized load balancing technique in cloud computing," *Int. J. Inf. Technol.*, vol. 15, no. 3, pp. 1357–1365, Mar. 2023.

[27] M. Ahmed, M. Khatri, F. Ahmed, and J. Goyal, "An optimized fuzzy-based load balancing in cloud computing," in *Proc. Int. Conf. Recent Adv. Electr., Electron. Digit. Healthcare Technol. (REEDCON)*, May 2023, pp. 323–328.

[28] R. Kumar and A. Chaturvedi, "Improved cuckoo search with artificial bee colony for efficient load balancing in cloud computing environment," in *Smart Innovations in Communication and Computational Sciences*. Singapore: Springer, 2021, pp. 123–131.

[29] R. Kumar, D. Bhardwaj, and R. Joshi, "Adaptive bat optimization algorithm for efficient load balancing in cloud computing environment," in *Advances in Computational Intelligence and Communication Technology*. Singapore: Springer, 2022, pp. 357–369.

[30] A. Pradhan, S. K. Bisoy, S. Kautish, M. B. Jasser, and A. W. Mohamed, "Intelligent decision-making of load balancing using deep reinforcement learning and parallel PSO in cloud environment," *IEEE Access*, vol. 10, pp. 76939–76952, 2022.

[31] D. Bagdasarov and E. I. Ostrovskii, "Reversion of chebyshev's inequality," *Theory Probab. Appl.*, vol. 40, no. 4, pp. 737–742, 1996.

[32] J. B. Sale, "The behaviour of the resting rock hyrax in relation to its environment," *Zoologica Africana*, vol. 5, no. 1, pp. 87–99, Jan. 1970.

[33] D. J. Druce, J. S. Brown, J. G. Castley, G. I. H. Kerley, B. P. Kotler, R. Slotow, and M. H. Knight, "Scale-dependent foraging costs: Habitat use by rock hyraxes (*Procavia capensis*) determined using giving-up densities," *Oikos*, vol. 115, no. 3, pp. 513–525, Dec. 2006.

[34] S. Badenhorst, K. L. van Niekerk, and C. S. Henshilwood, "Rock hyraxes (*Procavia capensis*) from middle stone age levels at blombos cave, South Africa," *Afr. Archaeolog. Rev.*, vol. 31, no. 1, pp. 25–43, Mar. 2014.

[35] D. Serruya and D. Eilam, "Stereotypies, compulsions, and normal behavior in the context of motor routines in the rock hyrax (*Procavia capensis*)," *Psychobiology*, vol. 24, no. 3, pp. 235–246, Sep. 1996.

[36] J. Zhou, T. Wang, P. Cong, P. Lu, T. Wei, and M. Chen, "Cost and makespan-aware workflow scheduling in hybrid clouds," *J. Syst. Archit.*, vol. 100, Nov. 2019, Art. no. 101631.

[37] N. J. Kansal and I. Chana, "Cloud load balancing techniques: A step towards green computing," *IJCSI Int. J. Comput. Sci. Issues*, vol. 9, no. 1, pp. 238–246, 2012.

[38] Z. Benlalia, A. Beni-Hssane, K. Abouelmehdi, and A. Ezati, "A new service broker algorithm optimizing the cost and response time for cloud computing," *Proc. Comput. Sci.*, vol. 151, pp. 992–997, Jan. 2019.

[39] A. Beloglazov and R. Buyya, "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers," in *Proc. MGC@ Middleware*, vol. 4, 2010, pp. 1–6.

[40] R. Buyya, A. Beloglazov, and J. Abawajy, "Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges," 2010, *arXiv:1006.0308*.

[41] N. George, A. B. Kadan, and V. P. Vijayan, "Multi-objective load balancing in cloud infrastructure through fuzzy based decision making and genetic algorithm based optimization," *IAES Int. J. Artif. Intell.*, vol. 12, no. 2, p. 678, Jun. 2023.

[42] B. Al-Khateeb, K. Ahmed, M. Mahmood, and D.-N. Le, "Rock hyraxes swarm optimization: A new nature-inspired metaheuristic optimization algorithm," *Comput., Mater. Continua*, vol. 68, no. 1, pp. 643–654, 2021.

SAURABH SINGHAL (Member, IEEE) received the Ph.D. degree, in 2021. He is currently an Assistant Professor with GLA University, Mathura. His research interests include resource management and placement optimization in the field of parallel and distributed systems.

ASHISH SHARMA is currently the Dean Academic Affairs with GLA University, Mathura. He has published more than 60 papers in international journals and conferences of repute. His research interests include software engineering, software testing, distributed systems, cloud computing, and artificial intelligence.

ANUSHREE is currently an Assistant Professor with GLA University, Mathura. She has published more than ten papers in international journals and conferences of repute. Her research interests include software engineering, software testing, and artificial intelligence.

PAWAN KUMAR VERMA received the Ph.D. degree from Lovely Professional University, Punjab, India. He has authored or coauthored many conferences and journals. His research interests include cloud computing and artificial intelligence.

MOHIT KUMAR (Member, IEEE) received the Ph.D. degree in CSE from Jaipur National University, Jaipur. He is currently an Associate Professor with the Department of IT, MIT Art, Design & Technology University, Maharashtra, India.

SAHIL VERMA (Senior Member, IEEE) is currently a Professor with SGT University, Haryana, India. He has authored or coauthored articles in reputed top-cited journals. He has chaired many sessions at international conferences.

KAVITA (Senior Member, IEEE) is currently a Professor with SGT University, Haryana, India. She has many research contributions in the area of cloud computing, the Internet of Things, and vehicular ad-hoc networks. Her research findings are published in top-cited journals.

MANINDER KAUR received the bachelor's and master's degrees in computer science from Guru Nanak Dev University, Amritsar, India. She is currently an Assistant Professor with the Department of Computer Science and Applications, Guru Gobind Singh College for Women, Chandigarh, India. She has 15 years of experience as an Assistant Professor. She has published many research articles. Her research interests include fog computing and the Internet of Things.

JOEL J. P. C. RODRIGUES (Fellow, IEEE) is currently with the College of Computer Science and Technology, China University of Petroleum, Qingdao, China, and the Senac Faculty of Ceará, Brazil; the Head of Research, Development, and Innovation; and a Senior Researcher with Instituto de Telecomunicações, Portugal. He is a member of the Internet Society, a Senior Member of ACM, and a fellow of AAIA.

RUBA ABU KHURMA received the bachelor's and master's degrees in computer science from Yarmouk University, in 2004 and 2007, respectively, and the Ph.D. degree in computer science from the University of Jordan, in 2021. She is currently an Assistant Professor with the Computer Science Department, Al-Ahliyya Amman University. She was a Researcher in cybersecurity and machine learning with the Department of Cybersecurity and Digital Forensics, Al-Balqa Applied University. She has published many articles in reputable journals, book chapters, and international conferences, such as CEC, ICPRAM, ECTA, ITISE, PICIT, and Evoapplications.

MARIBEL GARCÍA-ARENAS received the Ph.D. degree in computer science from the University of Granada, in 2003.

She was with the University of Jaén, from 2003 to 2007, teaching computer and telecommunication engineering degrees. She has been with the University of Granada, since 2007, where she is currently a Senior Lecturer with the Architecture and Technology of Computers Department, teaching electronic engineering and computer science degrees. She has worked on several European, national, and regional projects being the head researcher for three projects, two of them related to mobility and traffic at national and regional levels. She also has more than 20 publications in international journals and conferences. Her main research interests include evolutionary computation and high-performance computing.