International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 11-No.4 (2025) pp. 4619-4636
http://www.ijcesen.com

ISSN: 2149-9144

Copyright © IJCESEN

Research Article

A Fault-Tolerant and Load-Balanced VM Placement Approach for Cloud Computing Environments

Shubham Kumar[1], Isha Pathak Tripathi[2], Atul Tripathi[3]*

[1]Department of CS&IT, Mahatma Gandhi Central University, Bihar, India
Email: shubba2m@gmail.com- **ORCID:** 0000-0002-5247-0050

[2]Department of CSE, Indian Institute of Information Technology, Kota, India
Email: ish2a@gmail.com- **ORCID:** 0000-0002-5247-7990

[2]University School of Automation & Robotics, Guru Gobind Singh Indraprastha University, New Delhi, India
* **Corresponding Author Email:** atul.usar@ipu.ac.in- **ORCID:** 0000-0002-5247-9850

Article Info:

DOI: 10.22399/ijcesen.5237
Received : 05 October 2025
Revised : 25 November 2025
Accepted : 28 November 2025

Keywords (must be 3-5)

Virtual Machines,
Task Scheduling,
Quality of Service,
Differential Evolution,
Service Level Agreement

Abstract:

With the introduction of new technological advancements, the cloud environments become more complex day by day. Cloud Service Providers introduce unique, optimized approaches to deal with problems like load balancing and VM placement. In this research, a systematic approach has been proposed, defining the proper structural framework and functioning from the application layer to the physical layer, and that has shown a comparatively improved percentage of results in terms of time, energy, etc. An Alternative Differential Evolution-based Binding unique approach has been proposed for optimal Task-VM mapping by a systematic comparison of the solutions while ensuring efficient resource allocation and maintaining QOS requirements. In addition, an updated primary version and Threshold-based Secondary Version Scheduling are proposed for load balancing and backups, which play a vital role in fault tolerance with proper consideration of factors like cost and time. A Delay-Sensitive Dynamic Scheduling has also been proposed in the context of VM migration to deal with the defined aim of proper balance between nodes. The proposed approach shows more enhancements than the conventional and Max-Min approaches in task allocation and RAS performance. Similarly, regarding migration time and energy efficiency, DSDS (Delay-Sensitive Dynamic Scheduling) has improved results compared to the already defined ES-VSA and ADT-EC.

1. Introduction

The advancement of Industry 4.0 hinges on the seamless integration of the physical and virtual worlds, with cloud computing serving as a cornerstone for network virtualization. At its core, cloud computing relies on an extensive server infrastructure to deliver virtualized services to users over the network, establishing itself as a transformative business model. This model offers nearly unlimited capabilities and services on a scalable, pay-as-you-go basis. By providing a dynamic pool of virtualized computing resources via the network, cloud computing unifies distributed resources from various organizations and individuals. Users can effortlessly access diverse computing environments through network connectivity without needing to manage resource allocation or deployment intricacies [1]. In cloud computing, virtualization is essential since it consolidates computing resources from server clusters and dynamically assigns virtual machines to servers according to job demands. For each task execution request, a new virtual machine may be

instantiated, or the task may be allocated to an existing virtual machine associated with the same user [2].

The development of cloud computing, along with its widespread adoption and application, has led to significant growth in both the number and size of cloud data centers [3]. However, this growth has introduced challenges, particularly in terms of high energy consumption. Cloud data centers account for approximately 1.1% ~ 1.5% of global electricity consumption [4], making them energy-intensive entities. This trend has a substantial impact on global energy demands, necessitating thorough analysis. Widely cited data indicates that global energy usage by cloud data centers has doubled in the past decade [5]. Consequently, addressing energy consumption in cloud data centers has become a pressing issue of considerable interest [6].

In addition to energy consumption, there are too many issues that we have already discussed in the very first chapter of this thesis. One major key issue that is also taken into consideration in this research is the Quality of Service. Today, cloud systems must meet the QoS requirements of each user and application when delivering services. Therefore, users or applications must explicitly specify their QoS needs when requesting services from cloud resources. Furthermore, the degree to which QoS requirements are met is a critical metric for evaluating the effectiveness of task-scheduling strategies in cloud environments [7]. As a result, task scheduling strategies in cloud systems typically consider specific QoS objectives and the performance metrics of available resources to ensure that numerous application tasks can be efficiently matched with the available computing resources [8], [9]. There is a contract between a cloud service provider (CSP) and the user termed as Service Level Agreement (SLA). It specifies the expected QoS to be provided to the user. The SLA outlines key metrics such as throughput, reliability, blocking probability, and response time. It also details the payment processes and penalties for SLA violations. Given the critical role of SLAs in ensuring user satisfaction and system performance, the research on dynamic scheduling of virtual machine resources has garnered significant attention. The objective of this research is to reduce the overutilization of system resources while meeting the specified SLA standards, thereby delivering more efficient and stable services to users [10].

Although cloud computing has brought substantial growth opportunities to users and enterprises, small and medium-sized providers face significant challenges and competitive pressure from large, market-dominating cloud service providers [11]. To overcome these challenges, companies have adopted an effective strategy of forming cooperative relationships, pooling resources from multiple small- and medium-sized providers to establish a robust resource consortium known as a cloud federation. Cloud federations offer small- and medium-sized providers a viable means to achieve sustainable development in the highly competitive cloud computing market [12]. The issue of QoS management in cloud computing environments has become increasingly prominent. This problem encompasses a three-layer architecture, spanning from the user end (application layer) to the virtual end (virtual layer) and then to the physical end (physical layer). It involves two critical models: the task resource allocation model and the virtual resource allocation model. The three-layer resource model [4] of cloud computing is illustrated in Figure 1. In the three-layer resource model of cloud computing, the physical layer forms the foundation and comprises data centers housing a variety of hardware resources, including host servers. Positioned above the physical layer, the virtual layer leverages virtualization technology to partition physical hosts into virtual hosts, enabling more efficient and flexible resource management. At the top, the application layer delivers a broad spectrum of services to users, categorized into Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS), catering to diverse computing and business needs. The cloud computing model is categorized into two parts from a service standpoint. The Task allocation management module offers users an economical approach to distributing virtual resources. The second component is the virtual resource scheduling module, which creates a correlation between physical resources and virtual machines. This study posits that each host supplies a minimum of one virtual machine.

This study delves into the mechanisms of Task allocation in virtual machine management and how these tasks subsequently migrate to the physical resource layer. The research provides cloud service providers (CSPs) with flexibility in participating in resource sharing. At the same time, different CSPs within a cloud federation can act as both collaborators and competitors. Building on the models of cloud federations discussed in the literature [13], [14], [15], [16], this study aims to explore new avenues of cooperation for small- and medium-scale CSPs. For instance, in scenarios with significant fluctuations in resource demand (e.g., during network traffic peaks), CSPs with sufficient resource reserves are more inclined to adopt a collaborative approach, offering surplus resources

to partners. Conversely, when resources are limited, CSPs prioritize meeting the demands of their own users. The main contribution of this work is as follows:

1. The proposed research introduces a comprehensive approach to addressing the challenges of load balancing and fault tolerance in cloud computing environments.
2. Introduction of an innovative Task Allocation strategy leveraging the Alternative Differential Evolution (ADE) algorithm, specifically addressing multi-QoS task allocation problems.
3. Formulation of a virtual machine migration model incorporating cooperative and competitive dynamics to balance energy consumption and QoS.

2. Related Works

Khansari M et al. [17] have proposed a unique method named the Aware Genetic Algorithm First Fit (AGAFF) mainly combines a genetic algorithm with a first-fit approach. The goal is to optimize VM placement in cloud environments with various cloud data centers. The results shown through experiments prove that AGAFF has reduced energy usage by nearly 61% compared to the other four methods and has also reduced the number of physical machines needed.

Azizi S et al. [18] have proposed the Greedy Randomized Virtual Machine Placement (GRVMP) method, which uses the "power of two choices" model to place the corresponding Virtual Machines on physical servers that use less energy and are more efficient. By varying the performance metrics, the algorithm was tested with both simulated data and real-world cases, such as Amazon EC2. Experimental results thus obtained show that GRVMP reduces the use of power and avoids resource wastage better than other methods.

Tripathi et al. [19] discuss the challenge of efficiently placing Virtual Machines (VMs) in cloud data hubs. The authors have proposed a modified Dragonfly Algorithm (DA) inspired by the swarming behaviour of dragonflies to address this challenge. Simulations and comparisons have been carried out, which show that the modified DA performs better than existing methods by providing an effective solution for VM placement and helps optimize resource use in cloud environments.

Masdari M et al. [20] have introduced a unique method that combines a hybrid discrete multi-objective whale optimization algorithm enhanced using a multi-verse optimizer and improved using chaotic functions. The main goals are to manage resources through optimized VM placement and lower power use by minimizing the number of active physical machines. The method also avoids frequent VM migrations and, thus, the consumption of many resources as well. Comparing this algorithm with others, like the first fit, VMPACS, and MBFD, shows that it has shown a good reduction in power consumption and wastage in cloud data hubs.

Alboaneen D et al. [21] have suggested a metaheuristic approach to address the concept of combining the joint task scheduling and virtual machine placement (JTSVMP) problem. The algorithm mainly aims to map tasks to the virtual machines at the lowest possible execution cost with the constraint that it has to meet the deadline and place these VMs on the most utilized physical hosts (PHs) within capacity limits. The authors also compare this approach with traditional task-scheduling methods and other MOAs. Simulations and various experiments performed prove that combined optimization improves results by reducing execution cost, makespan, and imbalance while increasing the efficient use of physical hosts.

Singh D et al. [22] have presented a new method using the Enhanced-Modified Best Fit Decreasing (E-MBFD) method. This approach improves how VMs are placed on physical machines (PMs) to make cloud systems more efficient. It uses Artificial Neural Networks (ANN) to cross-check VM allocations, ensuring resources are appropriately utilized and detecting incorrect allocations. Results show that the E-MBFD method lowers power consumption and reduces breaches of the Service-Level Agreement (SLA) compared to older methods.

Gamal M et al. [23] have proposed a new method combining osmotic computing, based on osmosis in chemistry, with bio-inspired algorithms. This method is called the Osmotic Hybrid Artificial Bee and Ant Colony (OH_BAC) optimization algorithm. The specified algorithm combines the strengths of the artificial bee colony and the Ant Colony optimization techniques. It uses osmotic behaviour to automatically deploy and migrate virtual machines (VMs) across the cloud systems. This helps distribute the load more efficiently. The OH_BAC algorithm performs better than other bio-inspired methods.

Jena U et al. [24] proposed a new method that combines a modified version of the Particle Swarm Optimization (MPSO) algorithm with an improved version of the Q-learning algorithm. This combination, called QMPSO, is designed to balance loads among VMs dynamically. This improves VM performance, increases throughput, and reduces task waiting times. Simulations

validate the effectiveness of the QMPSO algorithm. The outcomes specify that QMPSO performs better than existing load balancing and scheduling methods. It balances the load more effectively, considers task priorities, and works well in simulated and real-world scenarios.

Junaid M et al. [25] introduced a new workload distribution and balancing algorithm called Data file type Formatting (DFTF). This algorithm combines a modified Cat Swarm Optimization (CSO) method with Support Vector Machines (SVM). This approach works by classifying different data types, such as images, texts, videos, and audio, using SVM classifiers. After classification, the load is distributed efficiently across virtual machines (VMs) using the modified CSO algorithm. The proposed method shows significant improvements compared to existing techniques. It improves throughput by a range of 7%, response time by about 8.2%, migration time by 13%, energy consumption is improved by 8.5%, optimization time is better than that by 9.7%, the overhead time is improved by 6.2%, of the SLA violations is better by 8.9%, and average execution time is moreover enhanced 9%.

Shafiq D et al. [26] have focused on solving workload balancing problems in cloud applications, especially within the Infrastructure as a Service (IaaS) model. The proposed load balancing (LB) algorithm aims to optimize resources and balance workloads effectively. It considers factors like quality of service (quality of service) parameters, VM priorities, and resource allocation. This method addresses gaps in previous research and shows significant improvements over the Dynamic LBA algorithm. It achieves an average resource utilization of 78%, reducing execution time and makespan. These results make it a more efficient option for cloud service providers to manage workloads in data centers.

Joshi N et al. [27] have proposed a new resource-sharing method that considers the priority levels of VMs. This ensures that resources are distributed fairly, preventing starvation and maintaining balanced load distribution. The technique also reduces waiting times for VMs needing more resources, improving overall system performance and throughput. The algorithm was tested on a prototype cloud system using OpenStack and CentOS Linux. The results showed a significant reduction in waiting times for overloaded VMs. The method successfully enables priority-based, starvation-free resource allocation. They also suggested that this approach could be extended to collaborative cloud computing systems, potentially enhancing load balancing across interconnected cloud environments.

Liu H et al. [28] presented a combined method for scheduling tasks and virtual machine placement (VMP) concerning energy-efficient data centers. The technique uses queueing theory and Deep Reinforcement Learning (DRL) to improve the scheduling of resources. The queueing theory model helps with task scheduling by reducing the number of virtual machines (VMs) needed while minimizing Service Level Agreement (SLA) violations. The DRL-based VMP algorithm selects physical hosts (PHs) for deploying VMs in real time. The simulation was performed on a small data center to evaluate this method. The results show that it keeps SLA violations low and balances resource usage better than other algorithms.

3. Proposed Model

First, the Task allocation module is comprehensively formalized with QoS equations, laying the foundation for subsequent explanations. Building on this, the architecture of the Task allocation process is described in detail, designed to mimic its core algorithmic design. Finally, the binding policy based on Alternative Differential evolution (ADEBBP) is introduced, which serves as a key component supporting the implementation of Task allocation during the process.

3.1 QoS Formula for Task Allocation Module

Quality of Service (QoS) in cloud services is determined using various attributes, as outlined in Table 1. Table 2 provides the meanings of the symbols utilized in the proposed model. The motivation for this research is drawn from the paper authored by Kun ma et al. [4]. They have proposed a Differential Evolution-based Multi-QoS-Target Scheduling Model (M-QoS-TSM). Our research work presents a novel task scheduling approach based on Multiple QoS, which addresses the task allocation challenge. This is mainly built upon three core QoS objectives: scheduling deadline, scheduling budget, and a composite parameter capturing the reliability, availability, and security performance.

The scheduling deadline establishes the maximum duration permitted for job execution across all virtual machines, whereas the scheduling budget delineates the overall expenditure for task completion inside the cloud environment. Reliability, availability, and security are essential differentiators among cloud service providers (CSPs), affecting user choices.

In the cloud environment, virtual resources are represented as a set of virtual machines, VM = {m| m ∈ 1, . . ., M}, where M is the total number of

VMs. Similarly, the set of tasks awaiting scheduling is denoted as T = {n| n ∈ 1, . . ., N}, where N is the total number of tasks. The model aims to optimize execution time and cost, constrained by deadline and budget. The multi-objective optimization model is defined as:

$$\min_{i,j}\begin{cases}\max_{n\in N}\sum_{m=1}^{n}\rho(m,n).\chi(m,n)\\ \text{subject to } \max_{n\in N}\sum_{m=1}^{M}\rho(m,n)\le TAT,\\ \sum_{m=1}^{M}\sum_{j=1}^{N}\zeta(m,n).\chi(m,n),\\ \text{subject to}\sum_{i=1}^{N}\sum_{n=1}^{N}\zeta(m,n)\le\tau\end{cases} \quad (1)$$

Here, TAT represents the execution deadline, τ is the total budget, ρ (*m, n*) is the execution time for task T_i on VM_j, and ζ (*m, n)* is the associated cost. The fitness function for task mapping is given as:

$$\gamma' = x\left(1-\frac{\rho(m,n)}{TAT}\right)+\phi\left(1-\frac{\varsigma(m,n)}{T}\right) \quad (2)$$

$$RAS\,(S)=\sum_{j=1}^{\lambda}(\varphi+\eta+\mu)\lambda \quad (3)$$

$$\theta=\begin{cases}\gamma'\cdot RAS(s), r'\neq 0\\ 0 \quad\quad , r'=0\end{cases} \quad (4)$$

The architecture depicted in Figure 2, illustrates the Task allocation process.

Step 1: Users submit tasks and quality of service requirements to the task scheduler, while VMs provide resource information.

Step 2: The scheduler generates simulation results based on the ADEBBP strategy.

Step 3: The scheduler evaluates the simulation results.

Step 4: If the VM meets the task requirements, the task is assigned; otherwise, the user is informed of the rejection.

Step 5: The VM executes the allocated task.

3.2 ADE-Based Binding Strategy

In scenarios with numerous people, identifying the ideal approach becomes challenging. The prevalent method involves utilizing diverse, intelligent optimization algorithms to approximate the perfect answer, which functions as an adequate resolution. The Alternative Differential Evolution (ADE) algorithm [29] is used to derive an approximation optimal solution. ADE enhances the standard Differential Evolution by incorporating four key modifications:

(1) a directed mutation strategy using the weighted difference between best and worst individuals, combined with standard mutation through a linear decreasing probability rule;

(2) two adaptive scaling factors as uniform random variables to improve population diversity;

(3) a dynamic non-linear increasing crossover probability to balance exploration and exploitation; and

(4) a hybrid mutation scheme combining random and modified BGA mutations to avoid stagnation

3.3 Algorithm 1: Pseudocode for applying ADE

Output: *The optimal solution (task scheduling result)*

1. *Initialize control parameters: CR_{min} =0.1, CR_{max}= 0.8, k= 4, $\rho = 1.0E-07$ (stagnation threshold), k= 75 (allowable stagnation generation), GEN= maximum number of generations*
2. *generate random initial population x_i^0 for i = 1 to NP*
3. *evaluate each solution f (x_i^0) for all i*
4. *identify the best solution x_b^0 and the worst solution x_w^0*
5. ***for** G= 0 to GEN -1 **do***
6. *CR= CR_{max} + (CR_{min} – CR_{max}) $\left(1-\frac{G}{GEN}\right)^k$*
7. ***for** i = 1 to NP **do***
8. ***if** (rand (0,1) ≥ $\left(1-\frac{G}{GEN}\right)$) **then***
9. *select random vector x_r^g where $r \neq i$*
10. *generate random vector f_l from (0, 1)*
11. $V_i^{(G+1)} = x_\gamma^G + f\left(x_b^G - x_\omega^G\right)$
12. ***else***
13. *select randomly r_1, r_2, r_3 from {1, 2, . . NP} where $r_1 \neq r_2 \neq r_3 \neq i$*
14. *generate random f_g from (-1, 0) U (0, 1)*
15. $V_i^{(G+1)} = x_{r_1}^G + f\left(x_{r_2}^G - x_{r_3}^G\right)$
16. ***end if***
17. *generate j_{rand} = random integer from (1, 2, . . D)*
18. ***for** (j= 1 to D) **do***
19. ***if** (random (0, 1) ≤ CR or j = j_{rand}) **then***
20. $u_{i,j}^{G+1} = v_{i,j}^{G+1}$
21. ***else***
22. $u_{i,j}^{G+1} = x_{i,j}^G$
23. ***end if***

24. ***end for***
25. ***if*** *($|f(x_i^G) - f(x_i^{G-1})| \leq \rho$ for k consecutive generations) AND(x_i^G is not the best)* ***then***
26. ***if*** *(random (0, 1) ≥ 0.5)* ***then***
27. *select random j_{rand} from {1, 2, . D}*
28. ***for*** *j= 1 to D* ***do***
29. ***if*** *(j= j_{rand})* ***then***
30. $u_{i,j}^{G+1} = a_j + random\ (0, 1).\ (b_j - a_j)$
31. ***end if***
32. ***end for***
33. ***else***
34. *Select random j_{rand} from {1, 2, . ,D}*
35. *compute α using the BGA distribution*
36. ***for*** *j= 1 to D* ***do***
37. ***if*** *(j= j_{rand})* ***then***
38. $u_{i,j}^{G+1} = x_{i,j}^G - random\ (0, 1).\ (b_j - a_j)\alpha$
39. ***end if***
40. ***end for***
41. ***end if***
42. ***end if***
43. *evaluate $f(u_i^{G+1})$*
44. ***if*** *$f(u_i^{G+1}) < f(x_i^G)$* ***then***
45. $x_i^{G+1} = x_i^G$
46. ***end if***
47. ***end for***
48. *Update x_b^{G+1} and x_w^{G+1} based on $f(x_i^{G+1})$*
49. ***end for***
50. ***return*** *the best solution*

3.4 Fault-Tolerant Scheduling Algorithm

This section designs a fault-tolerant scheduling algorithm (FSVC) for virtualized cloud platforms. The algorithm leverages backup version overlaps and virtual machine migration techniques to improve resource utilization while meeting fault tolerance requirements. When a new task arrives in the cloud, the system first schedules the primary version and then schedules the backup version. If either the primary or backup version is rejected, the task will be rejected.

Primary Version Scheduling Algorithm

Authors Ji Wang et al. [30] have proposed the Primary Version and Secondary version scheduling algorithms. Here the modified version of the same algorithm is used for primary version scheduling and secondary version scheduling. To leave sufficient time margins for the backup versions, the primary version should be completed as early as possible. Additionally, based on Theorem 1, the fundamental premise for two backup versions to overlap is that their primary versions are not on the same host. Therefore, primary versions cannot be concentrated on several hosts, as this would make overlapping backup versions infeasible and reduce resource utilization. Hence, the scheduling of primary versions must achieve the following two objectives.

1) The completion time of the primary version should be as early as possible.

2) The primary versions should be evenly scheduled across all active hosts.

One significant difference between cloud computing and traditional computing models is that tasks are scheduled on virtual machines rather than on hosts. However, to achieve the two objectives mentioned above, the state of the hosts must also be considered during scheduling.

Hosts with a smaller number of primary versions are preferred for task allocation. First, sort all the hosts in increasing order based on the number of primary versions scheduled on that. From all virtual machines on the hosts, choose one virtual machine where the completion time of the primary version is the earliest.

Through this strategy, the primary versions of newly arrived tasks will be prioritized for scheduling on hosts with fewer primary versions to achieve uniform distribution of the primary versions. If none of the existing virtual machines can accommodate the primary version $\boldsymbol{t_i^P}$, a new virtual machine can be created. The processing capability of the new virtual machine P_{new} must satisfy the following equation:

$$a_i + \frac{cs_i}{P_{new}} + delay = d_i \qquad (5)$$

Where delay represents the time reserved for starting a host and creating a virtual machine. The algorithm presented in this paper first attempts to place the new virtual machine on an active host. If no suitable active host is available, it performs virtual machine migration to consolidate existing virtual machines and free up host resources for the new virtual machine. If there are still no suitable active hosts, a host will be started from the powered-off hosts $H - H_a$, and the new virtual machine will be placed on the newly activated host. Algorithm 1 provides the pseudocode for scheduling the primary version in FSVC.

In Algorithm 2, virtual machine migration technology is incorporated to improve resource utilization. A two-phase strategy is employed again to ensure that while integrating virtual machines through migration, the distribution of primary versions becomes more uniform, and the impact on already scheduled versions is minimized as much as possible. First, select a host with the highest

number of primary versions. Migrate the virtual machines on that host to reduce the number of primary versions. Next, choose the virtual machine with the fewest primary versions on that host for migration, thereby minimizing the impact on already scheduled versions. The pseudocode for virtual machine migration is shown in Algorithm 3.

Algorithm 2: Pseudocode for updated Primary Version Scheduling

Require: *Set of hosts H_a, Task t_i^P with deadline d_i, Parameter α, Processing power P_{new}, Function vmMigration*

Ensure: *Assignment of t_i^P to a virtual machine or rejection*

1. *Based on the count of scheduled primary versions, Sort H_a in ascending order*
2. *Set find ← FALSE; EFT ← +∞; v ← NULL*
3. ***while*** *not all hosts in H_a have been scanned* ***do***
4. ***for*** *each h_k in H_a* ***do***
5. ***for*** *each v_{kl} in h_k.VmList* ***do***
6. *Calculate the earliest finish time $EFT_{kl}(t_i^P)$*
7. ***if*** *$EFT_{kl}(t_i^P) \leq d_i$* ***then***
8. *find ← TRUE*
9. ***if*** *$EFT_{kl}(t_i^P) <$ EFT* ***then***
10. *$EFT \leftarrow EFT_{kl}(t_i^P); v \leftarrow v_{kl}$*
11. ***else if*** *$EFT_{kl}(t_i^P) =$EFT* ***then***
12. ***If*** *resourceutilization (v_{kl}) < resourceutilization (v)* ***then***
13. *$v \leftarrow v_{kl}$*
14. ***end if***
15. ***end if***
16. ***end if***
17. ***end for***
18. ***end for***
19. ***end while***
20. ***if*** *find == FALSE* ***then***
21. *Create newVm with processing power P_{new}*
22. *bestHost ← NULL, maxResourceGap ← -1*
23. ***for*** *each h_k in H_a* ***do***
24. ***if*** *h_k can accommodate new Vm* ***then***
25. *resourceGap ← hostThreshold (h_k) – currentUtilization (h_k)*
26. ***if*** *resourceGap > maxResourceGap* ***then***
27. *maxResourceGap ← resourceGap*
28. *bestHost ← h_k*
29. ***end if***
30. ***end if***
31. ***end for***
32. ***if*** *bestHost ≠ NULL* ***then***
33. *Allocate newVm to bestHost*
34. *v ← newVm; find ← TRUE*
35. ***end if***
36. ***end for***
37. ***if*** *find == FALSE* ***then***
38. *sourceHost ← vmMigration (P_{new})*
39. ***if*** *sourceHost ≠ NULL* ***then***
40. *Allocate newVm to sourceHost*
41. *v ← newVm; find ← TRUE*
42. ***end if***
43. ***end if***
44. ***end if***
45. ***if*** *find == FALSE* ***then***
46. *Turn on a new host h_{new} from $H - H_a$*
47. ***if*** *MIPS of h_{new} satisfies P_{new}* ***then***
48. *Allocate new Vm to h_{new}*
49. *v ← newVm; find ← TRUE*
50. ***end if***
51. ***end if***
52. ***if*** *find == TRUE* ***then***
53. *Allocate t_i^P to v*
54. ***else***
55. *Reject t_i^P*
56. ***end if***

Now, discussing the time complexity of the primary version scheduling algorithm. Let N_a denote the total number of active hosts in the cloud platform, N_{vm} denote the maximum number of virtual machines on each active host, and N_{tw} denote the maximum number of tasks waiting to execute across all virtual machines. To determine a suitable virtual machine within the existing resources, the time complexity of lines 1 to 19 in Algorithm 2 in the best case (i.e., finding a suitable virtual machine in the first group of candidate hosts) is $O(\alpha N_a N_{vm} N_{tw})$, while in the worst case (i.e., scanning all active hosts), it is $O(N_a N_{vm} N_{tw})$. The time complexity of the function vmMigration (P_{new}) is $O(N_{vm} N_a^2)$, thus the time complexity of adding a new virtual machine in lines 20 to 56 is $O(N_a + N_{vm} N_a^2 + 1) = O(N_{vm} N_a^2)$. Therefore, in the best case, the time complexity of the primary version scheduling algorithm is $O(\alpha N_a N_{vm} N_{tw})$, while in the worst case, it is $O(N_a N_{vm} N_{tw} + N_{vm} N_a^2)$. It is worth noting that, typically, it is unnecessary to scan all hosts or add resources. Consequently, it can be observed that with the introduction of a two-stage strategy, this algorithm can achieve a more uniform distribution of primary versions while also reducing the algorithm's complexity.

Algorithm 3: Pseudocode for vmMigration Function

Require: *New processing power requirement* P_{new}
Ensure: *The source host sourceHost for VM migration*

for each host $h_i \in H_a$ ***do*** *// Calculate multi-factor host scores*
  Calculate host_score(h_i) = w1(primaries_count) + w2*(resource_utilization) + w3*(energy_efficiency)*
end for
Sort H_a *in decreasing order by host_score*
Initialize sourceHost ← NULL
for *each host* $h_i \in H_a$ ***do***
  for *each VM vj ∈ hi.vmList* ***do*** *// Calculate VM migration impact scores*
    Calculate vm_migration_score(v_j) = α(v_j.primaryTaskCount) + β*(v_j.resourceUsage) + γ*(v_j.migrationCost)*
  end for
  Sort h_i*.vmList in increasing order by vm_migration_score (prioritize low-impact migrations)*
  Initialize currentPower ← 0, migrationList ← empty list
  for *each VM* $v_j \in h_i$*.vmList* ***do***
    for *each host* $h_k \in$ *reversed(*H_a*)* ***do***
      if v_j *can be migrated to* h_k *and* h_k *satisfies VM constraints then*
        currentPower ← currentPower + v_j*.power*
        Add {v_j*,* h_k*} to migrationList*
        break
      end if
    end for
    if *currentPower* $\geq P_{new}$ ***then***
      Break
    end if
  end for
  if *currentPower* $\geq P_{new}$ ***then***
    Set sourceHost ← h_i
    Execute migrations in migrationList
    break
  end if
end for
return *sourceHost*

Secondary Version Scheduling Algorithm

Before designing the algorithm, we first introduce a concept described in reference [12], [30], known as the copy cost:

$$C_{kl}^{R}(t_i^B) = \frac{e_{kl}(t_i^B) - t_{kl}^{O}(t_i^B)}{e_{kl}(t_i^B)} \quad (6)$$

where $t_{kl}^{o}(t_i^B)$ denotes the computation time during which t_i^B overlaps with other secondary versions on virtual machine v_{kl}. The copy cost represents the ratio of the computation time occupied by this secondary version, excluding the overlapping portion, to the execution time. A low copy cost indicates that, apart from the overlapping section, this secondary version requires relatively little computation time. In other words, the actual resource consumption is minimal.

To improve resource utilization, two objectives must be achieved when scheduling secondary versions:

1) Secondary versions should adopt the passive mode as much as possible.
2) The copy cost of secondary versions should be minimized.

Based on these two objectives, this algorithm employs the ALAP strategy to minimize the copy cost of secondary versions while ensuring the latest possible start time for the secondary versions. Algorithm 4 presents the pseudocode for Threshold-based secondary version scheduling in FSVC.

Algorithm 4: Pseudocode for Threshold-Based Secondary Version Scheduling

INPUT: *Set of hosts* H_a*, Task backup version* t_i^B *with deadline* d_i*, Current host of primary version host (*t_i^P*), MIN_THRESHOLD (minimum acceptable utilization, i.e., 30%), MAX_THRESHOLD (maximum acceptable utilization, i.e., 80%)*

Output: *Assignment of* t_i^B *to a virtual machine or rejection*

Initialize eligibleHosts = NULL, find ← FALSE, LST ← +∞, C ← +∞, v ← NULL
for *each host* h_k *in {host (*t_i^P*))}* ***do***
  currentUtilization = (h_k*.totalMIPS -* h_k*.remainingMIPS) /* h_k*.totalMIPS * 100*
  If *currentUtilization ≥ MIN_THRESHOLD AND currentUtilization ≤ MAX_THRESHOLD* ***then***
    eligibleHosts.append(h_k*)*
  end if
end for
Sort eligibleHosts in increasing order by the remaining MIPS
for *each host* h_k *in eligibleHosts* ***do***
  for *each virtual machine* v_{kl} *in* h_k*.vmList* ***do***

11. Calculate the latest start time $LST_{kl}(t_i^B)$
*12. **if** $LST_{kl}(t_i^B) \leq d_i$ **then***
13. find←TRUE
14. Calculate the replication cost $CR_{kl}(t_i^B)$
*15. **if** $LST_{kl}(t_i^B) < LST$ or ($LST_{kl}(t_i^B) = LST$ and $CRkl(t_i^B) < C$) **then***
16. $LST \leftarrow LST_{kl}(t_i^B)$
17. $C \leftarrow CR_{kl}(t_i^B)$
18. $v \leftarrow v_{kl}$
19. end if
20. end if
21. end for
22. end for
*23. **if** find = TRUE **then***
24. Allocate t_i^B to v
25. else
26. Proceed with similar steps to lines 21 to 56 in Algorithm 2 (Primary Version Scheduling) but only consider hosts within the utilization thresholds
*27. **if** still no suitable VM **then***
28. Reject t_i^B
29. end if
30. end if

3.5 Virtual Machine Migration Model

Virtual machine migration typically occurs between the physical and virtual resource layers. The virtual resource layer, positioned above the physical layer, serves as the operational platform for the application layer, functioning as a virtual server. This layer can host multiple applications within a single virtual machine, creating a many-to-one relationship between applications and virtual machines. The physical resource layer, consisting of various hardware servers within the data center, provides the necessary resources to the virtual resource layer. This layer also follows a many-to-one mapping, where a single physical server can host multiple virtual machines.

Due to the evolving demands for application service access in the proposed architecture, a flexible cloud resource management platform is crucial. This platform must be able to modify the quantity of virtual machines to accommodate varying service demands. Scalability is essential, as it enables numerous migrations of virtual machines across hosts, influencing the overall usage of physical servers, the equilibrium of resource distribution, and the energy consumption of the data center.

1.1 Cloud Federation Model

In the field of cloud federation, cooperation models typically exhibit different cloud service providers (CSPs) working in harmony. First, they share resources, but each service provider maintains a clear and independent entity identity and upholds clear boundaries between each other. Second, the applications running within this large cloud federation are unaware of each other's specific locations, which is achieved through a virtual local area network (VLAN) that facilitates communication between different application components. Finally, these cloud service providers differentiate themselves based on factors such as their own costs and trust levels.

Cloud computing allows virtual machines allocated to user tasks to have different migration paths. These machines can locate themselves within the physical resources of the user's current CSP or within the physical resources of other CSPs. The flexibility of allocating virtual resources to physical domains leads to two possible scenarios: one is a cooperative model, where a user's virtual machines can freely migrate to any location; the other is a competitive model, where these virtual machines are restricted to migrating within the physical infrastructure of the original CSP. The cloud federation resource management framework is shown in Figure 3.

In the proposed model [4], the available resources of PM_v in CSP_x are represented by $R_{pn}(v_x)$, and the resource demand of VM_j during migration is represented by $D_{1n}(n_y)$. The cloud federation problem involves finding a mapping of each VM_j in the waiting queue to physical resources PM_v, which maximizes the utility function:

$$\begin{cases} \max\ D(m,v) = P(n_y, v_x)(R_{pn}(v) - D_{1n}(n)), \\ R_{pn}(j) \geq D_{vn}(v) \quad \forall n \in V, v \in P; \quad \forall n \in V, v \in P \rightarrow P(n_y, v_x \end{cases} \tag{7}$$

In this context, $P(n_y, v_x)$ is a binary parameter used in the model to distinguish between cooperative and competitive cloud computing, as shown in Equation (7).

$$P(n_y, v_x) = \begin{cases} 1, & if\ x \neq y, cooperative\ cloud\ federation \\ 0, & if\ x = y, competitve\ cloud\ federation \end{cases} \tag{8}$$

4. Delay-Sensitive Dynamic Scheduler (DSDS)

1. **Self-Evaluation of Hot Nodes:** PMs present in the data center will periodically check their status. If they are overutilized or underutilized, then they are marked as hot nodes. As soon as their utilization falls

in the acceptable range, then they will mark themselves as a normal node.

2. **Delay adaptive selection:** DSDS will calculate the migration time of all VMs of overutilized PMs to underutilized PMs. Then, the algorithm will select the set of VMs that will lead to the minimum migration time for under-utilized PM.
3. **Dirty page tracking and memory Delta-compression:** Virtual Machines transfer begins by placing the VM in inactive mode until migration completes. The movement of memory contents from the overloaded origin server to the destination in multiple steps. After the first phase, each phase handles only the pages that have undergone changes since the previous phase – these are named as "dirty pages." After reaching the final migration cycle, the source completely deactivates after sending the last modified memory segments. The system implements a delta-compression technique that only transmits the changed portions of dirty pages rather than complete pages, combined with a parallel transmission channel architecture that separates high-change-rate pages from stable memory regions. This mechanism is monitored continuously to track and move only the necessary memory elements rather than the whole data every time.
4. **PM Selection:** For a given set of VMs that need to be migrated, select a PM with minimum delay and also compare its resource requirement, i.e., CPU and memory with the available resources at the target PM. If the target node has a minimum delay and does not have enough resources, then another PM with the second minimum migration time is selected. The availability of enough resources is validated at every stage of selection.
5. **Resource Reservation:** Before initiating any migration, the scheduler temporarily reserves target resources on the destination PM to prevent race conditions when multiple migrations are triggered simultaneously.
6. **Virtual Machine Migration Log Update and Risk Handling:** After the completion of the process of migration, all the log tables and data get updated at both ends incorporating fault-resilience metrics that track partial migration failures and recovery actions, enabling the system to build an adaptive risk profile for different migration scenarios across heterogeneous infrastructure components.

4.1 Pseudocode for Delay-Sensitive Dynamic Scheduler

```
while True do
        // Self-evaluation of hot nodes
  overNodes ← {h ∈ Hosts | h.utilization > UPPER_THRESHOLD}
  underNodes ← {h ∈ Hosts | h.utilization < LOWER_THRESHOLD}
        // Delay adaptive selection
migrationPlan ← { }
for each host h ∈ overNodes do
   for each vm ∈ h.vms do
      targetHosts ← sortByMigrationTime(vm, underNodes)
      Add (vm, targetHosts) to migrationPlan
   end for
end for
        // PM Selection
finalPlan ← { }
for each (vm, targetList) ∈ migrationPlan do
   for each host ∈ targetList do
      if host.availableResources ≥ vm.requiredResources then
         Add (vm, host) to finalPlan
         break
      end if
   end for
end for
        //Resource reservation
for each (vm, targetHost) ∈ finalPlan do
   reserveResources(targetHost, vm.requiredResources)
end for
for each (vm, host) ∈ finalPlan do
   Execute migration
end for
for each (vm, host) ∈ finalPlan do
   updateMigrationLogs(vm, host)
end for
end while
```

5. Experimental Results

The experimental environment is CloudSim, which includes a complete cloud computing infrastructure supporting the configuration of the number and parameters of tasks, virtual machines, and physical machines in cloud computing simulation experiments. It also supports custom Task allocation strategies and virtual machine migration strategies. The CloudSim simulation environment used in the experiment is as follows: the CPU is an Intel I5, with a clock frequency of 2.6 GHz; the

memory is 16.0 GB; the hard disk drive (SSD) is 512 GB.

5.1 Task Allocation Management Model

The experiment utilizes eight compute nodes, each characterized by specific attribute parameters. These nodes are configured to emulate performance and computing capabilities modelled after the Amazon EC2 on-demand instance pricing framework. Table 3 presents the virtual machines used in the Task allocation management model, where MIPS indicates the CPU performance metric. The experiment simulated two tasks: Task P (Job P) and Task Q (Job Q), tested using three Task Qinding strategies: the ADEBBP strategy, the default CloudSim binding strategy (Conventional), and the Max-Min-based binding strategy. These strategies are designed to allocate tasks (Cloudlets) to Virtual Machines (VMs). Each task consisted of 20 Cloudlets requiring VM allocation. The details of the Task allocation management model are summarized in Table 4. For the simulations, the weight factors were set as $\chi = \phi$, with scheduling deadlines and budgets fixed at 100 and 1800, respectively.

The Alternative Differential Evolution (ADE) algorithm's operational parameters were configured as follows: a population size of 100, mutation scaling factor of 0.5, crossover rate of 0.1, and maximum iterations capped at 2000. The value range for the algorithm was bounded between -8 and 8. The simulation results for execution time for Job P and Job Q are presented figures 4 and 5 visualize the execution time and cost outcomes for both jobs, while the Resource Allocation Score (RAS) results are depicted in Figure 6- 7.

The experiment compared the performance of the ABDBBP with the CloudSim and Max-Min binding strategies for each job in terms of total execution time, total execution cost, and Resource Allocation Score (RAS).

From Figures 4- 5, it can be observed that for both Job P and Job Q, the execution time of Cloudlets in ADEBBP is shorter than that of the CloudSim binding strategy and the Max-Min binding strategy:

• For Job P, the total execution time of ADEBBP is shorter by 43.37% compared to the Conventional binding strategy and by 31.22% compared to the Max-Min binding strategy.

• For Job Q, the total execution time of ADEBBP is shorter by 50.2% compared to the Conventional binding strategy and by 37.76 % compared to the Max-Min binding strategy.

From Figures 6 and 7, it can be seen that:

• For Job P, the total RAS of ADEBBP is higher by 46.7% and 26.87% compared to the Conventional and Max-Min binding strategies, respectively.

• For Job Q, the total RAS of ADEBBP is higher by 48.28% and 29.61% compared to the Conventional and Max-Min binding strategies, respectively.

The simulation results show that the Conventional binding strategy in CloudSim does not optimize QoS, resulting in much lower QoS performance compared to the other two binding strategies. Compared to Max- Min, ADEBBP demonstrates significant advantages, which is better suited for resource allocation.

5.2 Implementation of DSDS

The experiment set up four types of virtual hosts, each consisting of 20 different virtual machines. These virtual machines are characterized by MIPS, Pes, RAM, bandwidth, and storage.

For VM migration, the experiment set up four types of virtual hosts, each consisting of 20 different virtual machines. These virtual machines are characterized by attributes such as MIPS, Pes, RAM, bandwidth, and storage size. Additionally, the experiment defined four CSPs, each with 10 hosts. The hosts were configured with parameters such as MIPS, Pes, RAM, bandwidth, and storage size. The maximum/minimum load thresholds for the virtual hosts were defined as 80%/0. The experimental parameters for VM migration taken from [4] are shown in Table 5.

The study also discusses energy consumption and SLA violations during VM migration and examines two VM migration algorithms: ES-VSA [31] and ADT-EC [32]. To validate the performance of DSDS, the experiment compared three performance metrics of the aforementioned migration strategies, namely SLA, VM migration time, and energy consumption. The results are shown in Figures 8, 9, and 10, respectively.

DSDS aims to avoid unnecessary VM migrations, giving it a significant advantage in VM migration time within cooperative and competitive coalitions. As shown in Figure 8, it reduces migration time by 84.05% and 80.89% compared to ES-VSA and by 86.20% and 80.02% compared to ADT-EC.

Furthermore, as illustrated in Figure 9, DSDS is the most effective technique, attaining minimal energy use in both cooperative and competitive coalitions. In comparison to ES-VSA, it decreases energy consumption by 22.99% and 32.18%, while relative to ADT-EC, it diminishes energy consumption by 23.72% and 28.52%.

In the proposed approach, the SLA correlates with the number of hot nodes, as DSDS aims to minimize superfluous VM migrations to conserve

energy. In comparison to ES-VSA and ADT-EC, its SLA violation rate is comparatively elevated. Figure 10 illustrates that the SLA violation rates for cooperative and competitive coalitions are elevated by 160% and 33.41%, as well as 19.16% and 29.31%, respectively.

Table 6 presents the comparative outcomes between cooperative and competitive coalitions, emphasizing the data and percentages for the superior-performing coalition. In terms of migration duration, competitive coalitions exhibit superior performance due to a reduced number of target node selection alternatives. Nevertheless, regarding energy consumption, cooperative coalitions exhibit superior performance as virtual machines possess a greater array of target node alternatives, albeit with the trade-off of broader migration scopes.

The Service Level Agreement (SLA) and energy performance have an inverse correlation. Consequently, competitive coalitions produce superior SLA results. The analysis indicates that the suggested DSDS aims to decrease the frequency of VM migrations, hence mitigating their effect on the overall performance of the cloud computing center. It accomplishes energy conservation while upholding the Service Level Agreement at an acceptable standard.

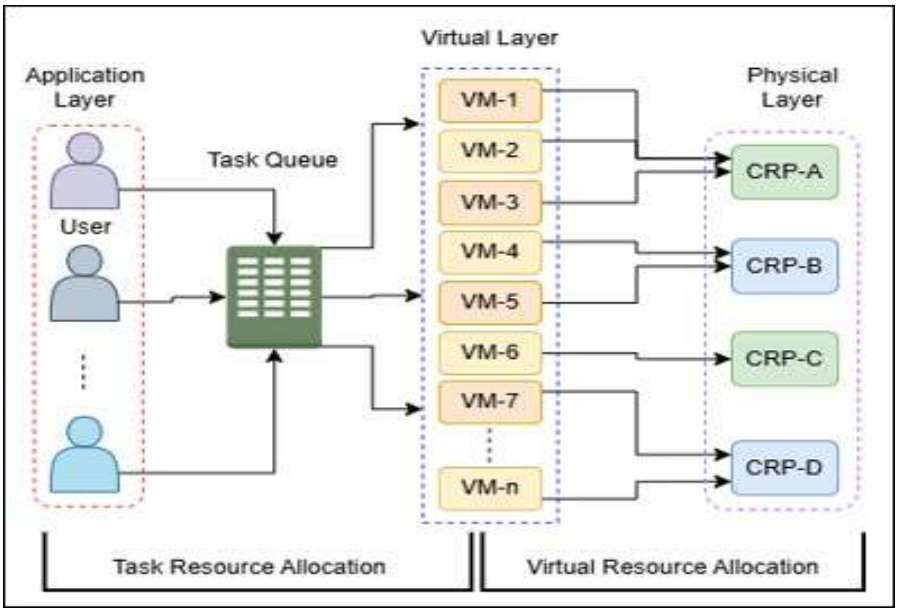


Figure 1: *Three- Layer Resource Model of Cloud Computing*

Table 1: *QoS Attributes*

Attribute Name	Meaning
Execution Cost (ζ)	Cost incurred by the CSP to execute a task
Execution Time (ρ)	Time required by the CSP to execute a task.
Reliability (φ)	Probability of the cloud service functioning normally.
Availability (η)	Likelihood of successful access to the cloud service.
Security (μ)	Security level offered by the cloud service.

Table 2: *Meaning of Symbols*

Symbol	Meaning
X	Mapping between tasks and VMs post-scheduling.
χ (m, n)	Boolean value indicating task i matches VM j.
γ′	Impact vector of execution time and cost for task-VM mappings.
$1-\left(\frac{\rho(m,n)}{TAT}\right)$	Time-saved score for task execution.

$1-\left(\frac{\varsigma(m,n)}{T}\right)$	Score related to cost saved upon task completion.
$RAS(s)$	Composite reliability, availability, and security score for system X.
φ, η, μ, λ	Reliability, availability, security of VM j, and the number of tasks it executes.
θ	Fitness function representing the score of mapping X.
χ, ϕ	Weight factors for execution time and cost importance.

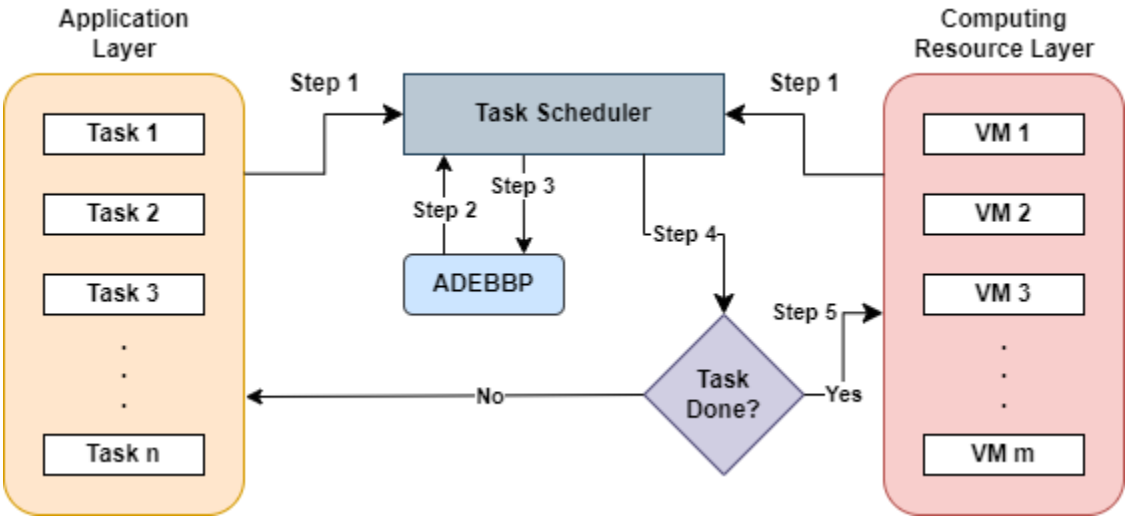


Figure 2: *Architecture for Task Allocation using ADEBBP*

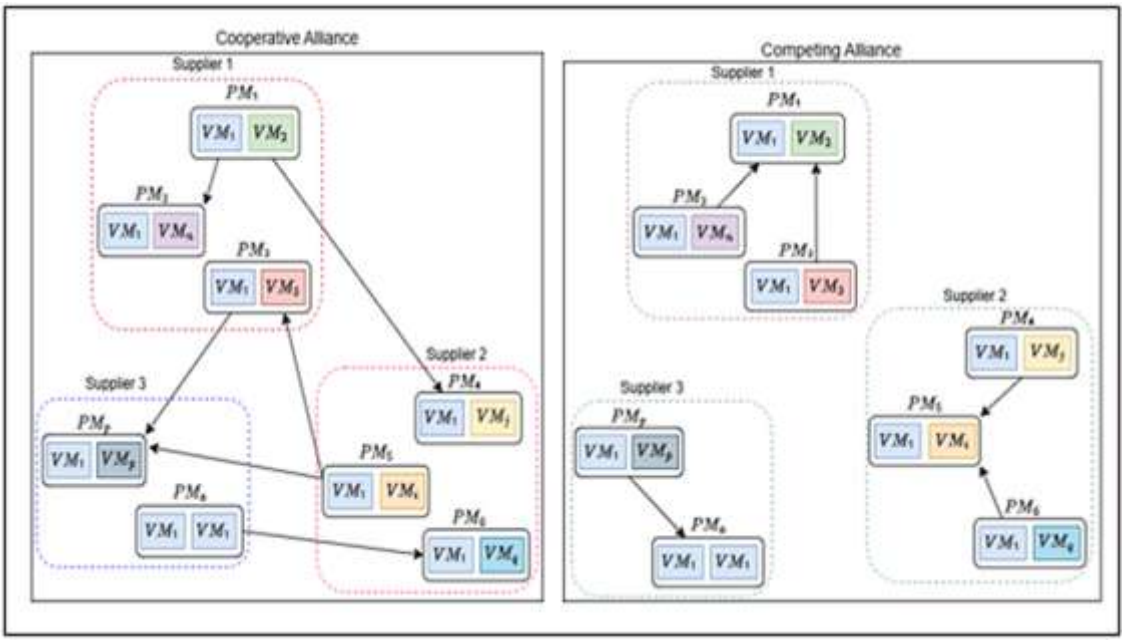


Figure 3: *Cooperative and Competitive Approaches*

Table 3: *Virtual Machines for the Task Allocation Management Model*

Property	VM-0	VM-1	VM-2	VM-3	VM-4	VM-5	VM-6	VM-7
MIPS	101	410	213	54	55	363	70	118
Availability (Ava)	40	10	20	20	30	50	10	30
Reliability (Rel)	10	10	10	60	20	40	10	20
Security (Sec)	20	40	10	40	10	60	50	30

Table 4: *Tasks of the Task Allocation Management Model*

CloudletID	Job P	Job Q
a	2040	950
b	720	130
c	190	330
d	120	1320
e	460	860
f	640	140
g	730	330
h	680	80
i	840	2240
j	840	310
k	1120	620
l	320	840
m	320	1170
n	60	690
o	1770	950
p	480	280
q	320	320
r	130	430
s	570	520
t	920	110

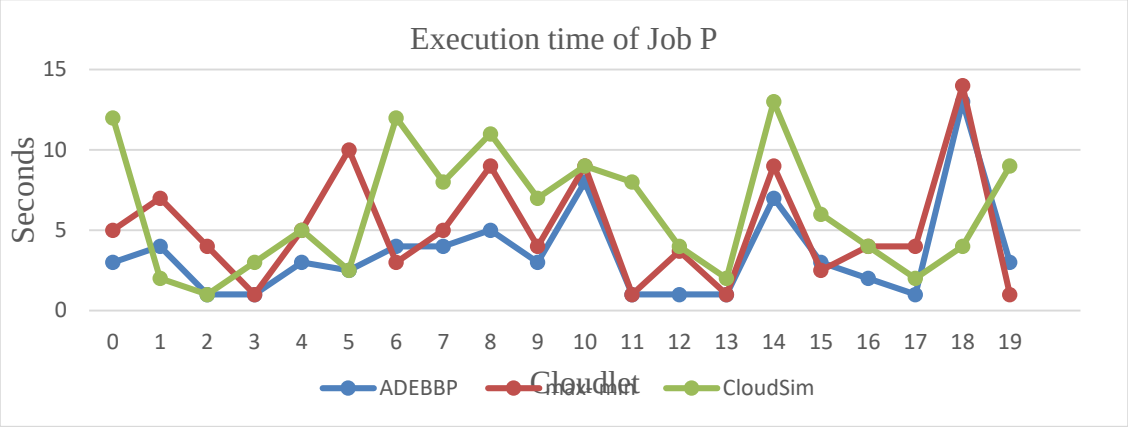


Figure 4: *Execution time of Job P*

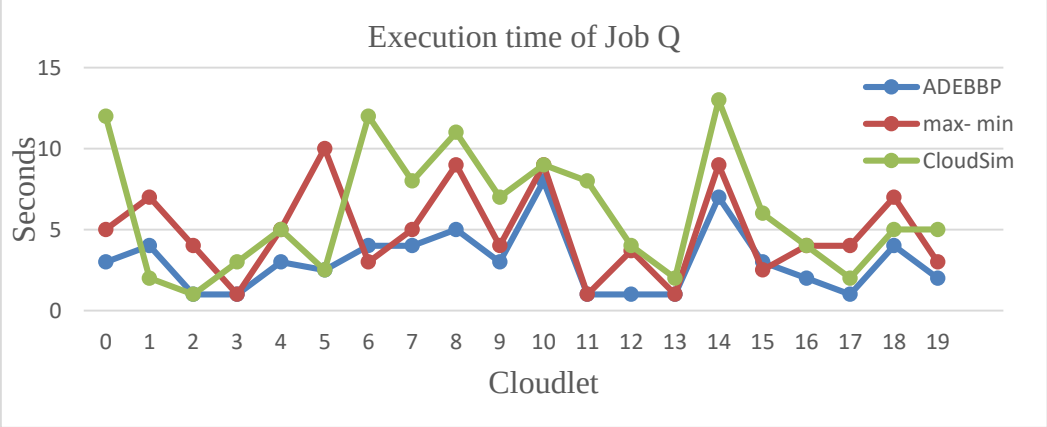


Figure 5: *Execution time of Job Q*

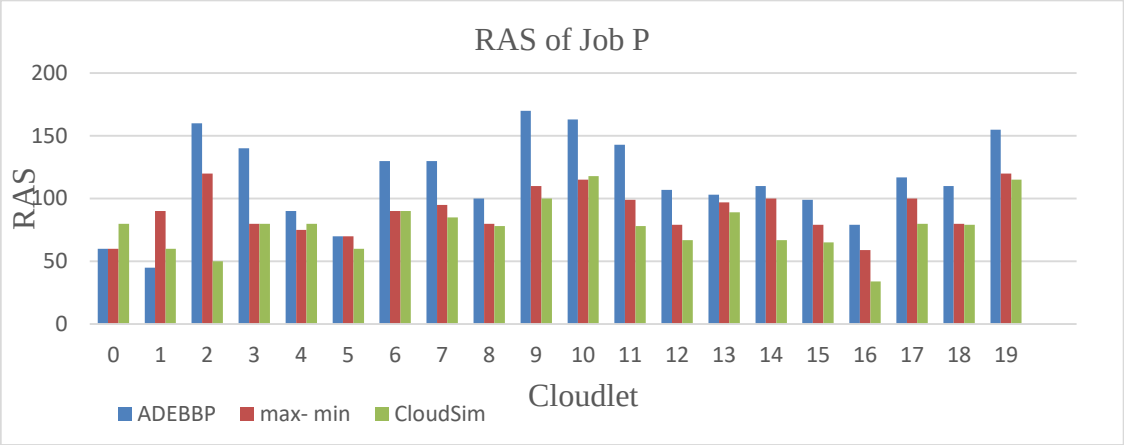


Figure 6: *RAS for Job P*

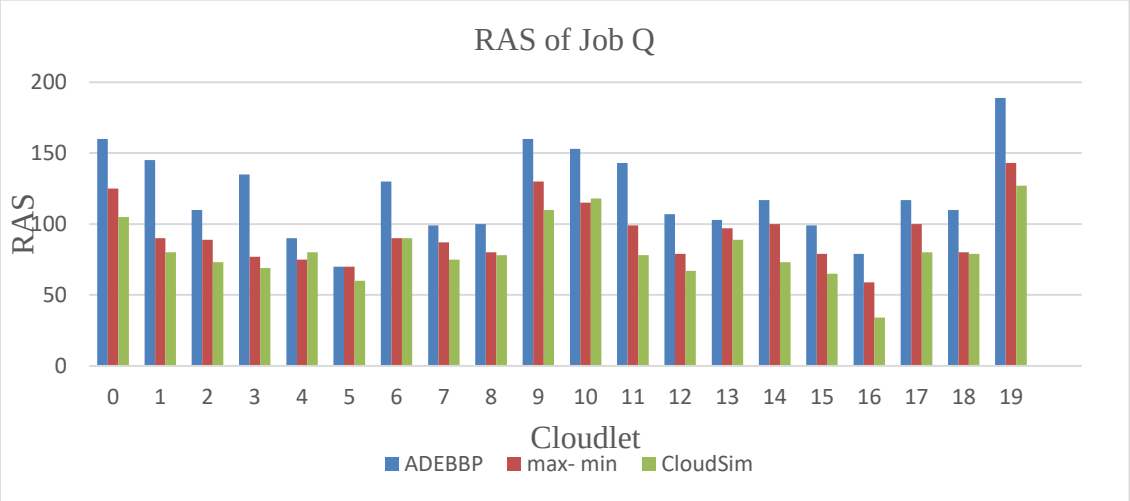


Figure 7: *RAS for Job Q*

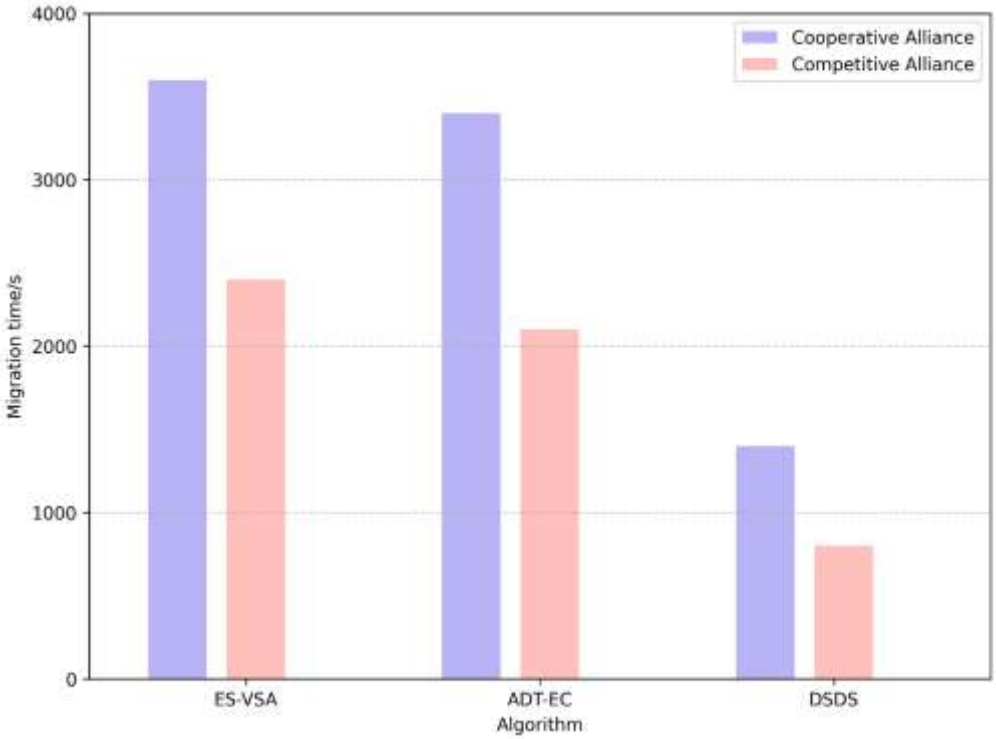


Figure 8: *Virtual Machine Migration Time Comparison*

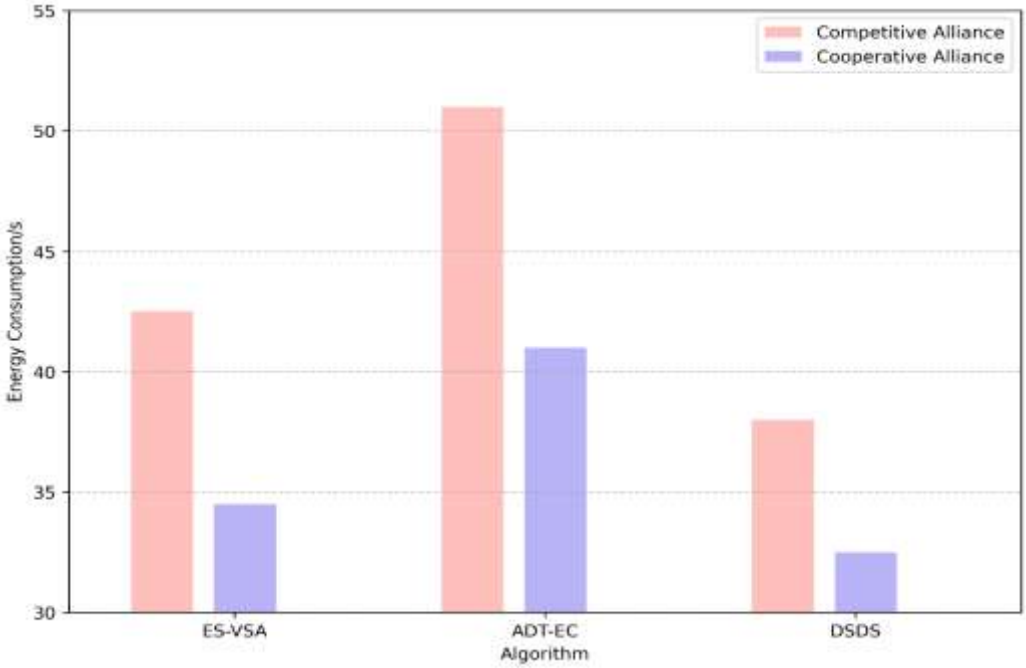


Figure 9: *Virtual Machine Migration Energy Consumption Comparison*

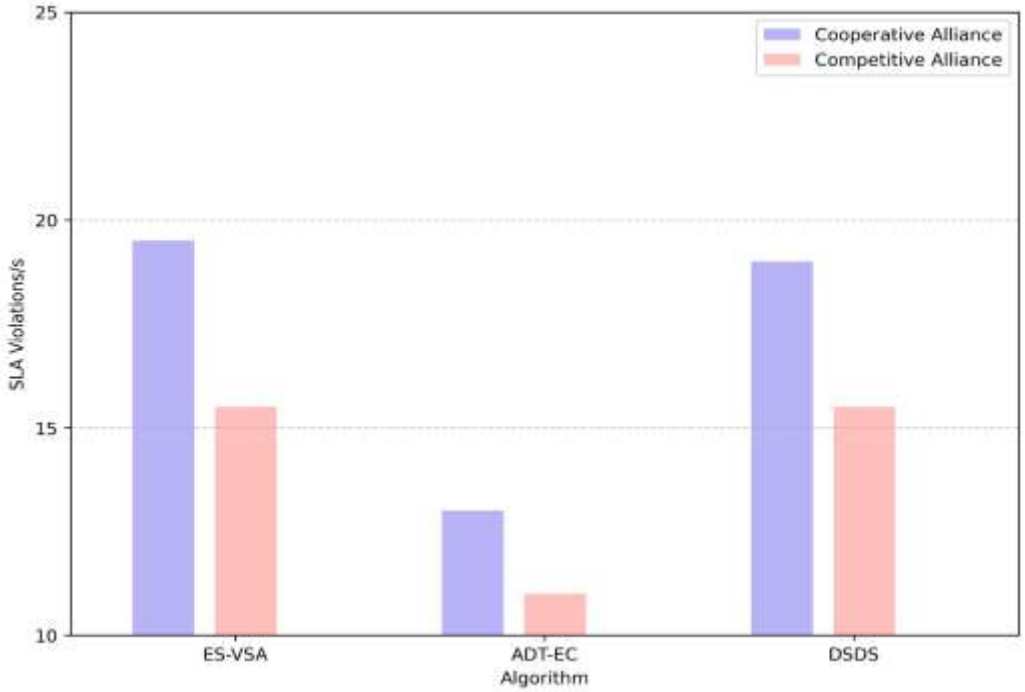


Figure 10: *Virtual Machine migration SLA comparison*

Table 4: *Experimental Parameters for VM Migration*

Type Settings		Quantity	MIPS	Pes	RAM	Size
VM Types	Type 1	20	500	1	870	2500
	Type 2	20	1000	1	1740	
	Type 3	20	2000	1	1740	
	Type 4	20	2500	1	613	
CSP 1	10 Hosts of Type 1	25	1860	2	4096	1000000
CSP 2	10 Hosts of Type 2	25	2660	2	4096	
CSP 3	10 Hosts of Type 3	25	2980	2	4096	
CSP 4	10 Hosts of Type 4	25	3220	2	4096	

Table 5: *Comparison of Cooperative alliances and Competitive alliances*

Comparison Items	Competitive	Cooperative	ES- VSA	ADT- EC	DSDS
Migration Time (Low)	✔	✖	45.46%	56.85%	36.85%

Energy Consumption (Low)	✖	✔	135%	19.7%	15.25%
SLA (Low)	✔	✖	7.52%	124%	11.63%

6. Conclusions

In cloud computing, where hardware resources are often limited, the quality of service (QoS) heavily relies on the efficiency of resource allocation strategies. This research addresses this using an enhanced version of algorithms for task allocation and load balancing. The approach introduces a Delay Sensitive Dynamic Scheduling-based cloud coalition model that is designed for small and medium-sized Cloud Service Providers (CSPs), which optimizes task allocation and virtual machine (VM) migration to achieve a balance between the overall benefits of the cloud coalition and the quality of quality of service. Future research could expand this work by incorporating external factors such as unreserved hosts, failure of hosts during the migration process, etc.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

References

[1] W. M. Danquah and D. T. Altilar, "Vehicular Cloud Resource Management, Issues and Challenges: A Survey," *IEEE Access*, vol. 8, pp. 180587–180607, 2020, doi: 10.1109/ACCESS.2020.3027637.

[2] P. Arthurs, L. Gillam, P. Krause, N. Wang, K. Halder, and A. Mouzakitis, "A Taxonomy and Survey of Edge Cloud Computing for Intelligent Transportation Systems and Connected Vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 6206–6221, Jul. 2022, doi: 10.1109/TITS.2021.3084396.

[3] C. Mahmoudi, F. Mourlin, and A. Battou, "Formal definition of edge computing: An emphasis on mobile cloud and IoT composition," in *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*, IEEE, Apr. 2018, pp. 34–42. doi: 10.1109/FMEC.2018.8364042.

[4] K. Ma, A. Bagula, and O. Ajayi, "Quality of Service (QoS) Modelling in Federated Cloud Computing," Nov. 2019, [Online]. Available: http://arxiv.org/abs/1911.03051

[5] J. Morel, V. Lebourgeois, J.-F. Martine, P. Todoroff, A. Begue, and M. Petit, "Recalibrating a sugarcane crop model using thermal infrared data," in *2013 IEEE International Geoscience and Remote Sensing Symposium - IGARSS*, IEEE, Jul. 2013, pp. 2806–2809. doi: 10.1109/IGARSS.2013.6723407.

[6] J. Luo, X. Li, and M. Chen, "Hybrid shuffled frog leaping algorithm for energy-efficient dynamic consolidation of virtual machines in cloud data centers," *Expert Syst Appl*, vol. 41, no. 13, pp. 5804–5816, Oct. 2014, doi: 10.1016/j.eswa.2014.03.039.

[7] K. Sellami, P. F. Tiako, L. Sellami, and R. Kassa, "Energy Efficient Workflow Scheduling of Cloud Services Using Chaotic Particle Swarm Optimization," in *2020 IEEE Green Technologies Conference(GreenTech)*, IEEE, Apr. 2020, pp. 74–79. doi: 10.1109/GreenTech46478.2020.9289818.

[8] J. Guo, Y. Li, C. Liu, Z. Zhao, and B. Zhang, "Research on a Virtual Machine Mode Transfer Method Supporting Energy Consumption Optimization," in *2022 24th International Conference on Advanced Communication Technology (ICACT)*, IEEE, Feb. 2022, pp. 77–80. doi: 10.23919/ICACT53585.2022.9728798.

[9] Z. Chen, L. Zhang, X. Wang, and K. Wang, "Cloud–edge collaboration task scheduling in cloud manufacturing: An attention-based deep reinforcement learning approach," *Comput Ind Eng*, vol. 177, p. 109053, Mar. 2023, doi: 10.1016/j.cie.2023.109053.

[10] M. K. Hussein and M. H. Mousa, "Efficient Task Offloading for IoT-Based Applications in Fog Computing Using Ant Colony Optimization," *IEEE Access*, vol. 8, pp. 37191–37201, 2020, doi: 10.1109/ACCESS.2020.2975741.

[11] G. Tricomi, G. Merlino, A. Panarello, and A. Puliafito, "Optimal Selection Techniques for Cloud Service Providers," *IEEE Access*, vol. 8, pp. 203591–203618, 2020, doi: 10.1109/ACCESS.2020.3035816.

[12] I. Petri, J. Diaz-Montes, M. Zou, T. Beach, O. Rana, and M. Parashar, "Market Models for

Federated Clouds," *IEEE Transactions on Cloud Computing*, vol. 3, no. 3, pp. 398–410, Jul. 2015, doi: 10.1109/TCC.2015.2415792.

[13] A. Bagula, M. Mandava, and H. Bagula, "A framework for healthcare support in the rural and low income areas of the developing world," *Journal of Network and Computer Applications*, vol. 120, pp. 17–29, Oct. 2018, doi: 10.1016/j.jnca.2018.06.010.

[14] Y. Ge, G. Zhang, M. N. Meqdad, and S. Chen, "A systematic and comprehensive review and investigation of intelligent IoT-based healthcare systems in rural societies and governments," *Artif Intell Med*, vol. 146, p. 102702, Dec. 2023, doi: 10.1016/j.artmed.2023.102702.

[15] S. Yan, L. He, J. Seo, and M. Lin, "Concurrent Healthcare Data Processing and Storage Framework Using Deep-Learning in Distributed Cloud Computing Environment," *IEEE Trans Industr Inform*, vol. 17, no. 4, pp. 2794–2801, Apr. 2021, doi: 10.1109/TII.2020.3006616.

[16] M. Fazio, A. Celesti, F. G. Marquez, A. Glikson, and M. Villari, "Exploiting the FIWARE cloud platform to develop a remote patient monitoring system," in *2015 IEEE Symposium on Computers and Communication (ISCC)*, IEEE, Jul. 2015, pp. 264–270. doi: 10.1109/ISCC.2015.7405526.

[17] S. M. Seyyedsalehi and M. Khansari, "Virtual Machine Placement Optimization for Big Data Applications in Cloud Computing," *IEEE Access*, vol. 10, pp. 96112–96127, 2022, doi: 10.1109/ACCESS.2022.3203057.

[18] S. Azizi, M. Shojafar, J. Abawajy, and R. Buyya, "GRVMP: A Greedy Randomized Algorithm for Virtual Machine Placement in Cloud Data Centers," *IEEE Syst J*, vol. 15, no. 2, pp. 2571–2582, Jun. 2021, doi: 10.1109/JSYST.2020.3002721.

[19] A. Tripathi, I. Pathak, and D. P. Vidyarthi, "Modified Dragonfly Algorithm for Optimal Virtual Machine Placement in Cloud Computing," *Journal of Network and Systems Management*, vol. 28, no. 4, pp. 1316–1342, Oct. 2020, doi: 10.1007/s10922-020-09538-9.

[20] S. Gharehpasha, M. Masdari, and A. Jafarian, "Virtual machine placement in cloud data centers using a hybrid multi-verse optimization algorithm," *Artif Intell Rev*, vol. 54, no. 3, pp. 2221–2257, Mar. 2021, doi: 10.1007/s10462-020-09903-9.

[21] D. Alboaneen, H. Tianfield, Y. Zhang, and B. Pranggono, "A metaheuristic method for joint task scheduling and virtual machine placement in cloud data centers," *Future Generation Computer Systems*, vol. 115, pp. 201–212, Feb. 2021, doi: 10.1016/j.future.2020.08.036.

[22] Shalu and D. Singh, "Artificial neural network-based virtual machine allocation in cloud computing," *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 24, no. 6, pp. 1739–1750, Aug. 2021, doi: 10.1080/09720529.2021.1878626.

[23] M. Gamal, R. Rizk, H. Mahdi, and B. E. Elnaghi, "Osmotic Bio-Inspired Load Balancing Algorithm in Cloud Computing," *IEEE Access*, vol. 7, pp. 42735–42744, 2019, doi: 10.1109/ACCESS.2019.2907615.

[24] U. K. Jena, P. K. Das, and M. R. Kabat, "Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 6, pp. 2332–2342, Jun. 2022, doi: 10.1016/j.jksuci.2020.01.012.

[25] M. Junaid *et al.*, "Modeling an Optimized Approach for Load Balancing in Cloud," *IEEE Access*, vol. 8, pp. 173208–173226, 2020, doi: 10.1109/ACCESS.2020.3024113.

[26] D. A. Shafiq, N. Z. Jhanjhi, A. Abdullah, and M. A. Alzain, "A Load Balancing Algorithm for the Data Centres to Optimize Cloud Computing Applications," *IEEE Access*, vol. 9, pp. 41731–41744, 2021, doi: 10.1109/ACCESS.2021.3065308.

[27] N. A. Joshi, "Technique for Balanced Load Balancing in Cloud Computing Environment," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 3, 2022, doi: 10.14569/IJACSA.2022.0130316.

[28] H. Liu, X. Zhou, K. Gao, and Y. Ju, "An integrated optimization method to task scheduling and VM placement for green datacenters," *Simul Model Pract Theory*, vol. 135, p. 102962, Sep. 2024, doi: 10.1016/j.simpat.2024.102962.

[29] A. W. Mohamed, H. Z. Sabry, and M. Khorshid, "An alternative differential evolution algorithm for global optimization," *J Adv Res*, vol. 3, no. 2, pp. 149–165, Apr. 2012, doi: 10.1016/j.jare.2011.06.004.

[30] J. Wang, W. Bao, X. Zhu, L. T. Yang, and Y. Xiang, "FESTAL: Fault-Tolerant Elastic Scheduling Algorithm for Real-Time Tasks in Virtualized Clouds," *IEEE Transactions on Computers*, vol. 64, no. 9, pp. 2545–2558, Sep. 2015, doi: 10.1109/TC.2014.2366751.

[31] Z. Xiao, Q. Qiu, L. Li, Y. Feng, Q. Lin, and Z. Ming, "An Efficient Service-Aware Virtual Machine Scheduling Approach Based on Multi-Objective Evolutionary Algorithm," *IEEE Trans Serv Comput*, vol. 17, no. 5, pp. 2027–2040, Sep. 2024, doi: 10.1109/TSC.2023.3341842.

[32] Z. Ma, D. Ma, M. Lv, and Y. Liu, "Virtual Machine Migration Techniques for Optimizing Energy Consumption in Cloud Data Centers," *IEEE Access*, vol. 11, pp. 86739–86753, 2023, doi: 10.1109/ACCESS.2023.3305268.