

Load Balancing in Cloud Computing based on Ant Colony Optimization and Crow Search Algorithm

Amar N. Alsheavi¹, Xingfu Wang¹, Wei Zhao², Ammar Hawbani³, Naji Alhusaini⁴, Ibrahim Abdulrab Ahmed⁵, and A.S. Ismail^{6,7}

¹School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China

²School of Computer Science and Technology, Anhui University of Technology, Anhui, China

³School of Computer Science, Shenyang Aerospace University, Shenyang 110136, China

⁴School of Computer and Information Engineering, Chuzhou University, Chuzhou, Anhui, 239000, China

⁵Computer Department, Applied College, Najran University, Najran 66462, Saudi Arabia

⁶College of Life Sciences and Oceanography, Shenzhen University, Shenzhen 518060, China

⁷Faculty of Science, Zagazig University, Zagazig 44519, Egypt

ammarnabil23050@gmail.com, wangxfu@ustc.edu.cn, zhaowei@ahut.edu.cn, anmande@ustc.edu.cn,

naji@chzu.edu.cn, iaalqubati@nu.edu.sa, a.sami@zu.edu.eg

Abstract—With the rapid increase in cloud services and the increasing shift toward them, balancing the cloud load has become a critical research issue. The increasing demand from customers for technology services worldwide is largely due to its direct impact on performance quality. Therefore, to provide better service quality, it is essential to consider reducing response time and cost in load balancing. This paper addresses the load-balancing problem by optimizing response time and computing cost in cloud computing systems. Specifically, we first formulated the load balancing problem in cloud computing mathematically by designing an objective function that optimizes response time and cost, including constraints to ensure task assignment to a single virtual machine (VM) and resource limits are not exceeded. To solve this optimization problem, we proposed a hybrid algorithm, ACOCSA, that combines Ant Colony Optimization (ACO) and Crow Search Algorithm (CSA). We implemented the ACOCSA algorithm, evaluated the response time, cost, and load balancing metrics, and found that our algorithm performed better in terms of response time and computational cost, and showed good fairness among VMs.

Keywords Load balance, Ant Colony Optimization (ACO), Crow Search Algorithm (CSA), Resource Management in the Cloud, Quality of Service (QoS), Cloud computing.

I. INTRODUCTION

Cloud computing is an evolving field focused on delivering services over the Internet. With the increasing demand for tasks and workloads, there has been a rise in load imbalance due to unequal distribution across nodes with varying capacities. This leads to some nodes being overloaded while others remain underutilized, affecting system performance. Advanced optimization methods, such as swarm intelligence and evolutionary algorithms, have shown promise in addressing these challenges by improving resource allocation and overall system performance [1], [2].

The Everything as a Service model in cloud computing has had a profound impact on both academia and industry by allowing organizations to move away from local data centers

to remotely managed data repositories [3], [4]. This shift helps reduce operational overhead while enhancing data accessibility and management [5]. Efficient resource utilization in cloud systems is essential for maintaining high productivity and performance.

Effective load balancing is critical to ensuring optimal resource use and maintaining system stability. Load balancing algorithms redistribute tasks from overloaded nodes to those with lighter loads, improving system performance, and preventing resource depletion, which can lead to overheating and increased carbon emissions [3]. The success of these algorithms is measured by metrics such as productivity, scalability, and energy efficiency [6].

There are two main types of load-balancing algorithms: static and dynamic. Static algorithms work well in stable and homogeneous environments, while dynamic algorithms are more suitable for cloud computing, where workloads and network traffic are constantly changing [7], [8].

Dynamic load balancing requires algorithms that adapt to real-time conditions, considering the capabilities of the nodes and the nature of tasks [9], [10]. Without efficient load balancing, users may experience delays or slow responses. By distributing tasks across multiple resources such as CPUs, memory, and network links, load balancing ensures optimal system performance and resource utilization [11].

Studies comparing different load-balancing algorithms have shown significant improvements in cloud efficiency, including better resource utilization, faster response times, and enhanced scalability [12]. This paper aims to address the challenges of achieving effective load balance in dynamic cloud environments, where load sizes and distributions change continuously [13]. Traditional algorithms often struggle with these dynamic shifts, leading to increased response times and decreased performance [14], [15].

Metaheuristic algorithms such as ACO and CSA offer

promising solutions to improve load distribution and reduce energy consumption. These methods are effective in managing dynamic workloads and enhancing the sustainability of cloud systems. ACO and CSA, known for their ability to explore and exploit the solution space, are well suited to the challenges posed by cloud data centers [16].

This paper contributes to the field by formulating the load-balancing problem in cloud computing through an objective function that optimizes response time and cost while adhering to resource constraints. We propose a hybrid algorithm, ACOCSA, that combines ACO and CSA, which effectively balances exploration and exploitation to achieve improved optimization results. The implementation and evaluation of the ACOCSA algorithm demonstrate its superior performance in reducing response time and computational costs, along with its scalability for large-scale cloud environments.

II. RELATED WORKS

Effective resource management in cloud computing is crucial to optimizing system load, performance, and user satisfaction. Various metaheuristic algorithms address load balancing challenges, such as optimizing resource allocation and minimizing response time and cost. For instance, Kishor et al. [17] proposed the LEWIS algorithm, which reduces latency and energy consumption. Sefati et al. [18] used Gray Wolf Optimization (GWO) for task distribution in cloud environments, showing its effectiveness in reducing costs and response times. Janakiraman and Priya [19] combined GWO with particle swarm optimization (PSO) for better resource allocation.

Thakur et al. [20] proposed RAFL, a hybrid algorithm that combines Phasor PSO and the Dragonfly Algorithm (DA) for optimized load balancing. Luo et al. [21] introduced a spatio-temporal method for energy cost optimization in distributed data centers. Recent advancements include hybrid approaches like the ANN-BPSO algorithm for task scheduling and load balancing [22]. However, many solutions still face issues with local optima and balance between response time and cost.

Despite these advancements, many solutions suffer from issues like getting stuck in local optima and a suboptimal balance between response time and cost. The proposed hybrid algorithm, ACOCSA, addresses these limitations by combining ACO and CSA, effectively balancing exploration and exploitation. This hybrid approach avoids local optima and leads to superior optimization results. Experimental results show that ACOCSA significantly reduces response time and computational cost compared to traditional methods, thus enhancing overall system performance and user satisfaction.

III. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we will look at the cloud computing system model, outline the task allocation problem, detail the constraints, and introduce the heuristic optimization algorithms we used.

A. Cloud Computing System Model

The CC system shown in Figure 1 includes users, request generators, and data center controllers, providing a load

balancing architecture for unbalanced cloud environments to achieve optimal load distribution. The model is based on a modified design by Gupta et al. [23], simplifying the roles of the VM manager and the monitor. The load balancing is performed at the physical machines (PM) level, focusing on distributing tasks within the same PM. The model includes intra-PM task migration, where tasks are moved between VMs within the same PM.

Users represent the end-clients who initiate requests for computing resources; these can be individuals or applications that need to process data or run applications on the cloud. The request generator acts as an intermediary component that formalizes these user requests, ensuring that they are in the appropriate format for processing by the data center. This separation helps to efficiently handle a large number of requests, as the request generator can batch similar requests, prioritize them, or perform initial processing to reduce the load on the data center. The request generator creates user requests that require computing resources. These tasks are managed by the data center controller, who directs the requests to the load balancer. The load balancer decides which VM should handle each user task. The load balancer distributes the workload among the VMs in that PM, ensuring optimal load distribution across the entire cloud infrastructure. This approach maximizes resource utilization and enhances the overall performance of the system.

Imagine a data center that has m resources and a user drops j workloads divided into n tasks. For the scheduling model, we are assuming the following [24], [25]:

- 1) Each task is its own thing.
- 2) Only one resource can be assigned a specific task.
- 3) There are more tasks than resources, as per this rule.
- 4) Keeping tabs on resource use in real time and guessing how long tasks will take are both doable.

Each user task is tagged with a unique flag called Cloud Letid, the processor count is dubbed Number of CPU, the file size sent to the data center is known as Input Size, the task's duration is called Length, and the output file size goes by Output Size.

The main activities in load balancing include identifying task requirements, checking the status of VM resources, scheduling tasks, allocating resources, and migrating tasks. Task requirements are identified and tasks are scheduled on suitable VMs using specific scheduling algorithms. Resource allocation means assigning the resources needed to the scheduled tasks using resource allocation policies. The success of a load balancing algorithm depends on the efficiency of the scheduling algorithm and the allocation policy [26]. Migration is a key step in load balancing, involving two types: VM migration and task migration. VM migration moves a VM from one physical host to another to avoid overload, while task migration moves tasks between VMs. Research shows that task migration is more time and cost effective than VM migration, leading to a shift from VM migration to task migration to achieve better load balancing in cloud environments [27].

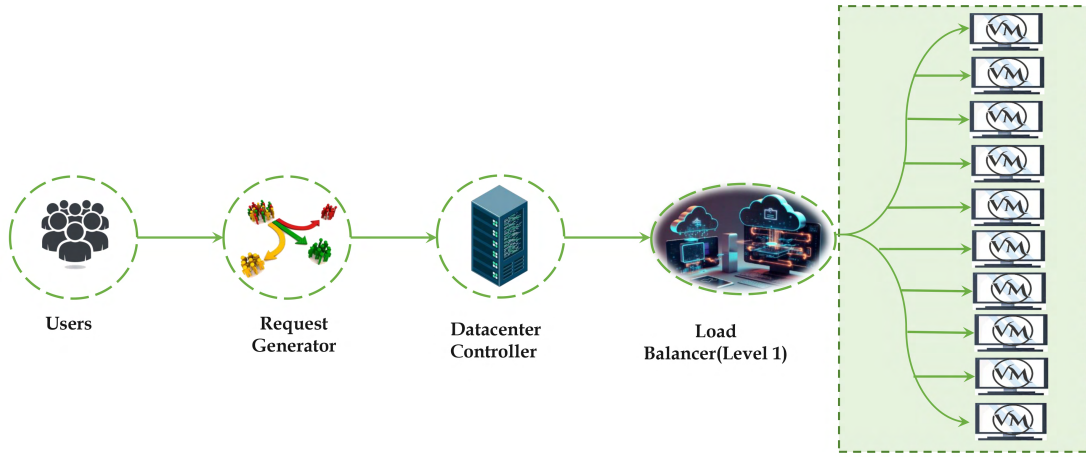


Figure 1: Load balancing architecture

B. Problem Formulation

In this subsection, we provide an overview of the assumptions and notations, each symbol is clearly defined with its corresponding meaning to maintain clarity and consistency throughout the document.

We assume that the cloud environment consists of users $U = \{u_1, u_2, \dots, u_k\}$. Each user u_i generates a set of tasks $M_i = \{t_{i1}, t_{i2}, \dots, t_{ij}\}$, where t_{ij} represents the task j created by the user u_i . The set of available VMs is denoted by $N = \{v_1, v_2, \dots, v_n\}$. Each VM has certain capacities (C_{cpu_k}), representing the maximum amount of computational power (in CPU units) that v_k can provide, determining how many and how quickly tasks can be processed by this VM. The cost refers to the computational expense involved in executing tasks within a cloud computing environment. This primarily includes factors such as CPU usage, memory consumption, and the total time required to complete VM tasks. The aim is to minimize these costs to improve resource allocation and overall system performance.

The expected response time T_{ijk} to execute task t_{ij} on VM v_k is formulated in (1), including the processing time, transmission time, and propagation time.

$$T_{ijk} = T_{process} + T_{trans} + T_{prop} \quad (1)$$

The resources required to execute task t_{ij} on v_k are indicated by R_{ijk} , which includes the CPU resources needed by task t_{ij} when executed on v_k . In this paper, we use the M/M/1 queuing theory to estimate the average waiting and processing times $T_{process}$ of tasks in VMs, as formulated in (2), where λ is the expected arrival rate and μ is the expected service rate. These values are estimated based on the observed patterns of task arrival and processing during the simulation. In our scenario, we suppose that both transmission and propagation times are minimal. This simplification aligns with our research focus on optimizing load balancing within VMs on the same physical machine.

$$T_{process} = \frac{1}{\mu - \lambda} \quad (2)$$

The load on a VM L_k represents the total resource requirements for all tasks assigned to it.

$$L_k = \sum_{i=1}^{|U|} \sum_{j=1}^{|M_i|} R_{ijk} \quad (3)$$

The load deviation D_k and the auxiliary variable S_k are used to formulate the load balancing objective, ensuring that the contribution of each VM to the objective function is always positive, thus enhancing the load balancing.

$$D_k = \bar{L} - L_k, \forall k \quad (4)$$

$$S_k = |D_k| \quad (5)$$

The $\bar{L}$ in (4) is calculated according to (6). These equations are used in the constraints to account for both negative and positive load deviations, leading to a better balance in task distribution.

$$\bar{L} = \frac{1}{N} \sum_{k=1}^{|N|} L_k \quad (6)$$

Each user generates a set of tasks that require computing resources for execution. Our goal is to distribute these tasks across the available VMs in a way that ensures load balancing, improves resource utilization, and reduces response time.

Our objective function is linear and aims to minimize three main aspects, response time, cost, and load balancing. It is represented by the following equation:

$$\begin{aligned} \min Z = & \alpha \sum_{i=1}^{|U|} \sum_{j=1}^{|M_i|} \sum_{k=1}^{|N|} T_{ijk} x_{ijk} + \beta \sum_{i=1}^{|U|} \sum_{j=1}^{|M_i|} \sum_{k=1}^{|N|} C_{ijk} x_{ijk} \\ & + \gamma \sum_{k=1}^{|N|} S_k \end{aligned} \quad (7)$$

$$\text{subject to } \sum_{k=1}^{|N|} x_{ijk} = 1, \forall j, \forall i \quad (8)$$

$$\sum_{i=1}^{|U|} \sum_{j=1}^{|M_i|} R_{cpu_{ijk}} x_{ijk} \leq C_{cpu_k}, \forall k \quad (9)$$

$$x_{ijk} \in \{0, 1\} \quad (10)$$

$$C_{ijk} = R_{ijk} \times C_{res} \quad (11)$$

To determine the appropriate values for the coefficients α , β , and γ , we consider the primary goals of minimizing the response time and cost while achieving the load balance. Since response time and cost are of higher priority, we assign higher values to α and β because they directly impact user experience and operational cost, and a relatively lower value to γ to ensure fair task distribution, so that $\alpha + \beta + \gamma = 1$. These coefficients can be adjusted on the basis of experimental results to optimize the performance of the hybrid algorithm. By integrating these constraints, we ensure that tasks are allocated in a way that improves resource utilization while adhering to the capacity limits of each VM.

The VM placement problem considered in this research is subject to the following equality and inequality constraints:

- 1) Single Assignment Constraint: Each t_{ij} must be assigned to exactly one VM, Constraint (8). This constraint ensures that each task is assigned and executed by one single VM, preventing duplication in assignment.
- 2) Resource constraints: The total resources required by the tasks assigned to any VM must not exceed its available resources, constraint (9).

To solve this problem, we use heuristic optimization algorithms such as ACO and CSA. These algorithms explore the search space and iteratively improve solutions to find the optimal one. The basic steps to use these algorithms include the following:

- Initialization: Randomly initialize solutions to cover a wide search space.
- Iterative improvement: Improve solutions through multiple iterations using techniques such as evaporation and cooperation to enhance the quality of the solution over time.
- Solution Update: Update the solution values based on performance to guide the search towards the most effective solutions.

C. Algorithm Design

To keep things running smoothly in a real-world system and achieve the perfect load balance, it is crucial to think about how well the equipment (we are talking about node resources here) is performing.

Figure 2 presents the ACO process flow. It outlines steps from initializing the system with VM tasks to finding the best solution through iterative pheromone updates, as shown in

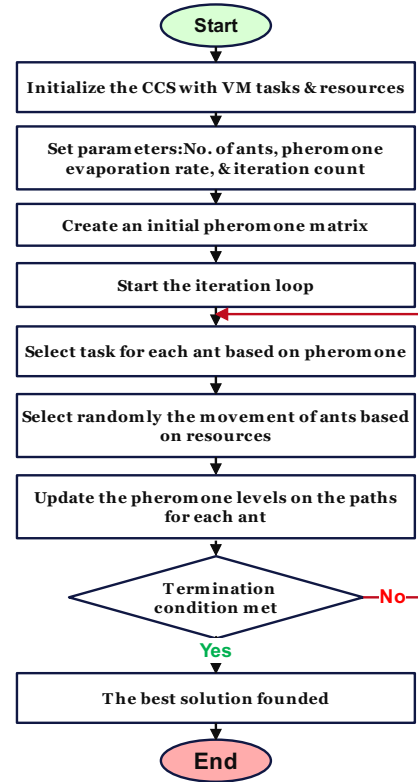


Figure 2: Architecture of the proposed method of ACO

Figure 3, which illustrates the CSA flow chart. This diagram shows the sequence from initializing crows' positions to optimizing their paths for effective load balancing.

Through this process, the job scheduling model and the ACO algorithm are integrated to achieve efficient job scheduling in the cloud computing environment, with the aim of achieving a better balance in resource loading and reducing the total cost, thus enhancing efficiency and effectiveness in cloud resource management.

So, what sets the algorithms proposed in this study apart from others is that they factor in the expected finish time for each task of each VM into the waiting task queue. Based on this parameter, the algorithm picks the VM with the least expected completion time and the lowest utilization rate for the task distribution. In the proposed method, specific factors such as the shortest route, the best cost, and the minimum response time are also taken into account.

This approach aims at enhancing resource distribution and task scheduling efficiency within the cloud computing environment, enabling better performance and quicker responses to user requests while making sure we use resources in the best way possible. By integrating algorithms such as Ant Colony and Crow Search while considering QoS criteria, this method offers a comprehensive solution for load management and task allocation in complex cloud systems.

D. Experiment

The proposed approach was developed and put to work in a MATLAB R2019a environment. We used MATLAB to do

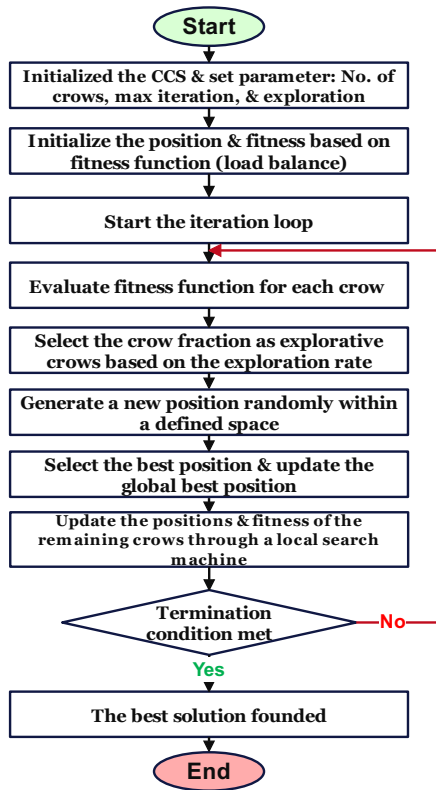


Figure 3: Architecture of the proposed method of CSA

the calculations. The algorithms were run on a computer with an Intel Core i5 processor at 2.20 GHz, 8 GB of RAM, and Windows 10. To see how well our suggested algorithm works, we used the ACO and CSO algorithms to solve the problem. We compare the ACOCSA with the original ACO and CSA, as well as with MLVM, PAVM, IACO, GAA, and HGAACO.

The simulation used the following parameters: 100 tasks (num tasks), 10 virtual machines (num VMs), task load (task load) with random values between 0 and 100 million instructions per second (MIPS), VM capacity (VM capacity) with random values between 0 and 100 MIPS, and task arrival rate (task arrival rate) with random values between 0 and 10 tasks per second. Also, the key parameters of the CSA were set with an awareness probability of 0.1 and a flight length of 2. In addition, the number of ants (20) and crows (20) was chosen to effectively implement the ACOCSA algorithm. These values were selected based on experimental trials to enhance solution exploration and resource utilization. The parameters were fine-tuned through extensive testing to achieve optimal performance to reduce response time and computational cost.

The ACOCSA algorithm, which combines the ACO and CSA algorithms, was used to achieve an overall improvement in task distribution and resource management in the cloud environment.

The experimental steps included constructing solutions for each ant in the ACO by assigning tasks to VMs based on pheromone levels and VM capacity and then updating the

pheromone levels based on the quality of the solutions found by the ants. In the CSA, new positions for the crows were generated and their fitness evaluated, with the best positions updated based on the probability of awareness and flight length. The ACO and CSA results were combined to determine the optimal overall task allocation.

Time Complexity Analysis: The time complexity mainly depends on I_{max} , q , M , and N . The samples to be considered during updates include calculations of response time and cost for each task-VM pair, along with pheromone matrix updates. Spatial complexity analysis focuses on the memory required to store the pheromone matrix and the ants' search tables, and it depends on the number of M and the number of N . The optimized loop structure in the second algorithm reduces the overall time complexity, making it more efficient in handling large workloads, thereby improving performance and resource utilization more effectively.

E. Results and Analysis

In this section, we provide a comprehensive analysis of the performance of the hybrid ACOCSA algorithm compared to various commonly used algorithms in cloud resource management. Specific algorithms were selected in certain figures and tables based on analytical objectives and scientific reasons to offer clear and effective comparisons. These selections cover various approaches in resource and task management, highlighting ACOCSA's efficiency across different scenarios.

It is important to note that a limited number of algorithms were selected for each comparison to avoid unnecessary complexity and ensure clarity of results, as explained in the description of each figure and table below. This approach ensures a focused analysis that improves the understanding of the improvements offered by the hybrid ACOCSA algorithm.

Figure 4 compares the ACO and CSA algorithms. These two algorithms were specifically chosen for several scientific reasons related to the analytical objectives of this research. The ACO algorithm represents the fundamental algorithm on which the hybrid ACOCSA algorithm is based, allowing a direct comparison to determine the improvements brought about by integrating CSA. On the other hand, the CSA protocol is one of the evolutionary algorithms that relies on different methodologies for resource management optimization. This figure shows variations in the performance of both ACO and CSA as the number of tasks increases, highlighting the key differences in their performance. These results suggest that combining ACO and CSA into a hybrid algorithm can provide significant performance improvements, demonstrating the superiority of the proposed solution in this study over individual algorithms. This confirms the cost-effectiveness and adaptability of the proposed hybrid algorithm in resource management.

Figure 5 illustrates the response times for the ACO and ACOCSA algorithms. The decision to compare these two algorithms comes from the need to highlight the improvements achieved by the hybrid ACOCSA algorithm over its foundational algorithm, ACO, so that we can effectively demonstrate the performance enhancements, particularly in terms of

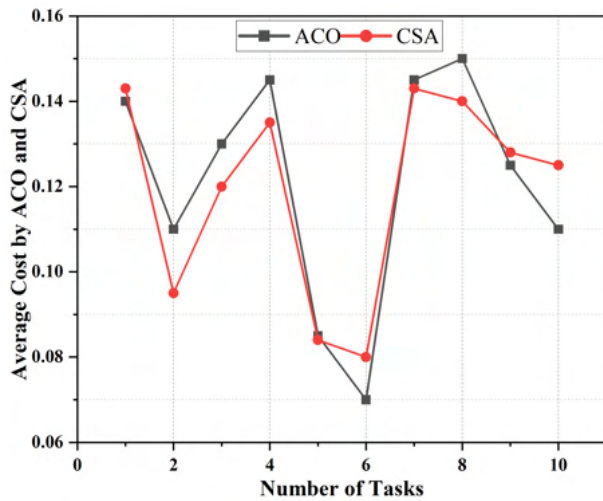


Figure 4: Average cost by ACO and CSA

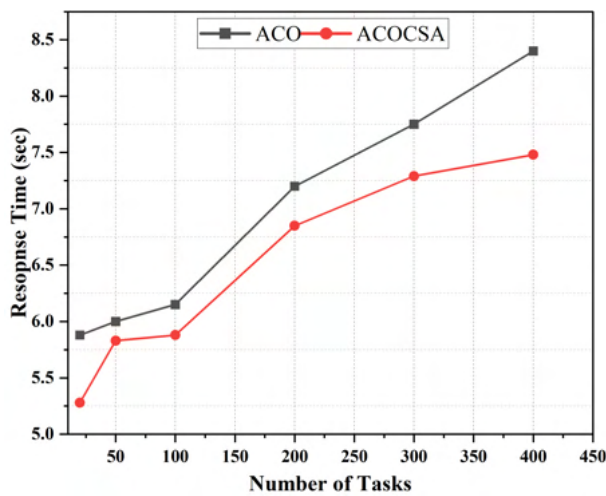


Figure 5: The average of response time

handling larger workloads more efficiently. This comparison focuses on showcasing how the integration of ACO with CSA in the hybrid algorithm leads to shorter response times as the number of tasks increases.

Figure 6 presents a comparative analysis of the average response time for eight algorithms (ACOCSA, GIJA, AOA, RSA, DMOA, ACO, MLVM, PAVM) across various task sizes. These algorithms were selected to represent a diverse set of resource management strategies and optimization techniques, providing a comprehensive view of each algorithm's effectiveness in reducing response time. ACOCSA is based on hybrid techniques that combine traditional and modern strategies, allowing it to outperform many of the other algorithms, particularly in larger task scenarios. ACOCSA demonstrated superior efficiency compared to the other algorithms, consistently achieving shorter response times as the task size increased. This highlights its high capacity to handle large workloads effectively and significantly reduce response time. For example, when managing 400 tasks, ACOCSA showed a

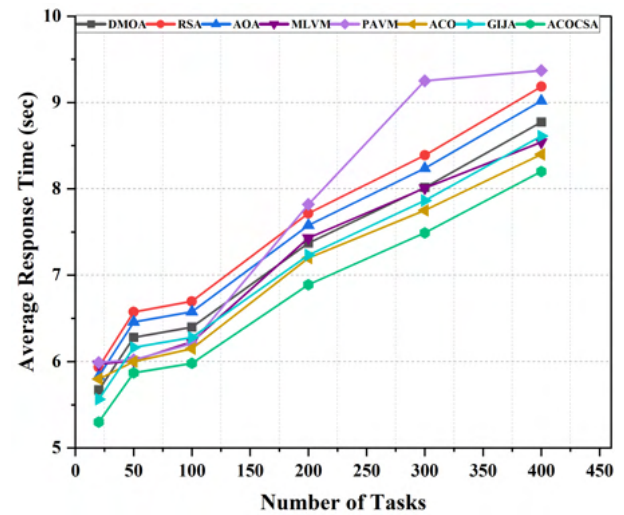


Figure 6: Comparative of average response time with the previous algorithms

clear advantage over algorithms such as MLVM and PAVM, which rely on different task management techniques (such as machine learning and task prioritization strategies). ACOCSA was able to reduce the response time to 8.20 seconds, while MLVM and PAVM had significantly higher response times for the same number of tasks.

Although GIJA demonstrated strong performance, it could not consistently match ACOCSA in reducing response time. For example, at 400 tasks, GIJA had a response time of 8.61 seconds, which is significantly higher than ACOCSA's 8.20 seconds, which shows ACOCSA's clear advantage in handling larger task volumes more efficiently. As for the RSA, AOA, and DMOA algorithms, they showed stable performance but higher average response times across all tasks compared to ACOCSA, reinforcing the hypothesis that ACOCSA is the most efficient algorithm in this context. This highlights ACOCSA's superiority in managing large workloads efficiently, reducing time costs, and improving response times across different task sizes, making it a scalable and highly effective solution for complex optimization problems in dynamic environments.

Figure 7 offers a comprehensive comparison of the Average Completion Time for different algorithms, namely: ACOCSA, GIJA, AOA, RSA, DMOA, ACO, MLVM, PAVM, and IACO, in varying task sizes ranging from 20 to 100 tasks. ACOCSA demonstrates a clear advantage in all task sizes compared to the other algorithms. For example, when handling 100 tasks, ACOCSA completes the tasks in approximately 46 seconds, while GIJA and AOA take 51 and 53 seconds, respectively. This difference highlights ACOCSA's superior ability to optimize task processing efficiency and reduce completion time. Although GIJA performed well in reducing task completion times, it couldn't consistently match ACOCSA's performance, especially in larger task sizes. AOA showed a performance level close to that of GIJA, but it also failed to achieve the efficiency level achieved by ACOCSA. This emphasizes that

ACOCSA's hybrid approach, which combines traditional and modern techniques, offers a significant advantage.

In general, ACOCSA proves to be the most efficient and responsive algorithm among the group, making it an excellent choice for applications that require effective task management, particularly when dealing with larger task volumes. Furthermore, the results underscore ACOCSA's capability to excel in environments that demand high efficiency in processing complex and large-scale tasks. This outstanding performance establishes ACOCSA as a reliable algorithm for improving resource management and minimizing task completion times, positioning it as an ideal solution for applications requiring scalability and high performance in dynamic and evolving environments.

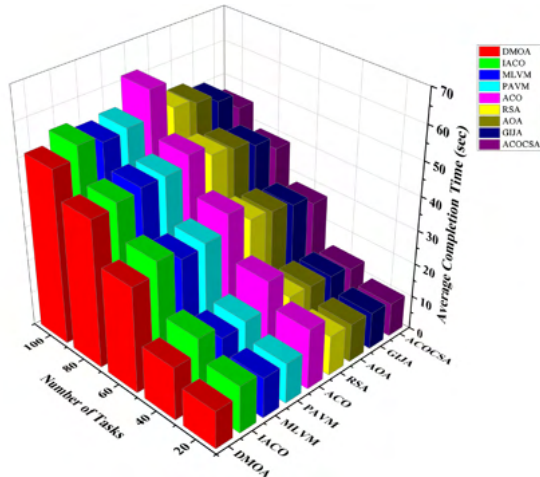


Figure 7: Average completion time

Table I provides a comprehensive comparison of task completion times (task time spans) across ten different algorithms: ACOCSA, ACO, GAA, HGAACO, MLVM, PAVM, DMOA, RSA, AOA and GIJA, for varying task sizes ranging from 10 to 100 tasks. This table highlights the differences in the algorithms' performance in reducing task completion times, offering valuable insights into the efficiency of each algorithm in optimizing resource usage and managing tasks effectively.

ACOCSEA consistently demonstrates superior performance across all task sizes compared to the other algorithms. For example, when handling 100 tasks, ACOCSA completes them in just 300 seconds, while other algorithms such as GAA and PAVM require significantly more time, taking 410 and 330 seconds, respectively. This indicates the high efficiency of ACOCSA in managing large task volumes and reducing completion times, making it a highly suitable option for applications that require fast processing and high efficiency. As for the other algorithms, HGAACO and GIJA show average results, with HGAACO performing better than ACO and GAA, but ACOCSA still outperforms them in task completion efficiency. DMOA and RSA also produce acceptable results, but fall behind ACOCSA in terms of overall performance.

Overall, this table emphasizes the clear advantage of ACOCSA in improving task completion times across various

task sizes. This strong performance underscores the importance of ACOCSA as a reliable and powerful algorithm for addressing complex optimization problems that demand efficient task management while minimizing completion time, making it an ideal solution for dynamic and fast-paced environments.

TABLE I: Comparative analysis of task time spans: ACOCSA across different optimization algorithms

Algorithms	Number of Tasks					
	10	20	40	60	80	100
ACO	80	190	240	350	400	450
GAA	90	170	230	340	390	410
HGAACO	70	180	220	290	320	340
MLVM	80	180	220	340	385	405
PAVM	60	185	210	290	320	340
DMOA	66	174	215	237	325	330
RSA	69	182	224	248	339	345
AOA	67	177	218	241	330	336
GIJA	65	171	211	232	318	324
ACOCSEA	60	158	195	215	295	300

The ACOCSA algorithm achieved significant improvements in response time and computational cost compared to traditional algorithms. By combining the efficiency of ACO with the resource utilization capabilities of CSA, ACOCSA was able to improve task distribution across available resources, reducing the response time by 2-3%. Furthermore, the integration of ACOs exploration and CSAs resource exploitation allows ACOCSA to significantly improve system performance and reduce computational costs through optimal task allocation and more efficient use of resources. This combination indicates a reduction in resource consumption and overall system improvement.

Finally, we calculate the fairness in load distribution among virtual machines using the optimized ACOCSA algorithm. Figure 8 shows a significant improvement in the fairness of the load distribution among virtual machines using the optimized ACOCSA algorithm over 100 iterations. The system starts with a low fairness of approximately 86 and gradually increases to approximately 89.5, indicating a substantial improvement in load balance. This improvement means that the algorithms used were effective in distributing loads more evenly among the virtual machines, enhancing the efficiency and performance of the distributed computing system. The final fairness value reflects a more equitable distribution of the loads.

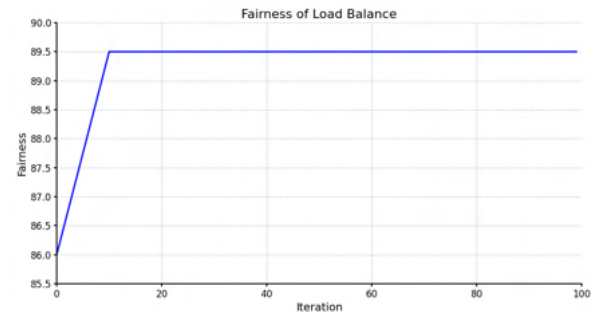


Figure 8: Fairness enhancement in load balancing among VMs with optimized ACOCSA algorithm

IV. CONCLUSION

This paper addresses optimization issues related to load balancing that interfere with the overall performance of systems. The research suggests using a combined method of the ACO algorithm and the CSA to control balancing the workload in cloud computing environments to make the whole load scheduling thing work better. The ACOCSA is able to manage task scheduling based on the current state info of the nodes. To check how well this proposed algorithm works, the results were put side by side with some other standard algorithms for comparison. It became clear that the designed algorithm holds its ground according to various performance criteria, which shows that the proposed method does a better job of keeping things balanced in a changing environment. This study focuses on response time and cost as performance metrics, without considering other QoS factors such as reliability and security. Furthermore, implementing the hybrid algorithm requires significant computational resources, which may limit its use in resource-constrained environments. Also performance of the algorithm depends on carefully tuning parameters such as awareness probability and flight length in CSA, and the number of ants in ACO. Future research should consider integrating other QoS metrics and exploring adaptive mechanisms that allow the algorithm to dynamically adjust to changing cloud environments, improving its efficiency and practical application.

ACKNOWLEDGMENTS

This work was supported by Open Fund of Anhui Engineering Research Center for Intelligent Applications and Security of Industrial Internet, under Grant IASII24-04. Also, the project Elderly Home Environment Safety Monitoring and Evaluation Technology and Application (No. 2022AH051102) funded by the Anhui Education Department provided essential resources and support. Amar N. Alsheavi is grateful for financial support from the ANSO scholarship. Amar expresses sincere gratitude to Hongyi Xie for his invaluable guidance, insightful discussions, and support throughout the development of this research.

REFERENCES

- [1] D. Chauhan, R. Cheng *et al.*, "Competitive swarm optimizer: A decade survey," *Swarm and Evolutionary Computation*, vol. 87, p. 101543, 2024.
- [2] N. Hogade and S. Pasricha, "A survey on machine learning for geo-distributed cloud data center management," *IEEE Transactions on Sustainable Computing*, vol. 8, no. 1, pp. 15–31, 2022.
- [3] H. Singh, S. Tyagi, and P. Kumar, "Cloud resource mapping through crow search inspired metaheuristic load balancing technique," *Computers & Electrical Engineering*, vol. 93, p. 107221, 2021.
- [4] T. Chakraborty, A. N. Toosi, and C. Kopp, "Elastic power utilization in sustainable micro cloud data centers," *IEEE Transactions on Sustainable Computing*, vol. 8, no. 3, pp. 465–478, 2023.
- [5] S. Chakraborty, A. K. Saha, and A. Chhabra, "Improving whale optimization algorithm with elite strategy and its application to engineering-design and cloud task scheduling problems," *Cognitive Computation*, vol. 15, no. 5, pp. 1497–1525, 2023.
- [6] L. Cheng, Y. Wang, F. Cheng, C. Liu, Z. Zhao, and Y. Wang, "A deep reinforcement learning-based preemptive approach for cost-aware cloud job scheduling," *IEEE Transactions on Sustainable Computing*, 2023.
- [7] L. Abualigah and A. Diabat, "A novel hybrid antlion optimization algorithm for multi-objective task scheduling problems in cloud computing environments," *Cluster Computing*, vol. 24, no. 1, pp. 205–223, 2021.
- [8] M. P. Garkuwa, R. Mustapha, and K. Haruna, "An improved crow search load balancing algorithm using threshold-based strategy for virtual machine allocation," 2023.
- [9] D. A. Shafiq, N. Jhanjhi, and A. Abdullah, "Load balancing techniques in cloud computing environment: A review," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 7, pp. 3910–3933, 2022.
- [10] A. Alsheavi, A. Hawbani, W. Othman, X. Wang, G. Qaid, L. Zhao, A. Al-Dubai, L. Zhi, A. Ismail, R. Jhaveri *et al.*, "Iot authentication protocols: Challenges, and comparative analysis," *ACM Computing Surveys*, 2024.
- [11] D. Tennakoon, M. Chowdhury, and T. H. Luan, "Cloud-based load balancing using double q-learning for improved quality of service," *Wireless Networks*, pp. 1–8, 2023.
- [12] M. Gaharwar and D. Dhukkarya, "X-band multilayer stacked microstrip antenna using novel electromagnetic band-gap structures," *IETE Journal of Research*, vol. 69, no. 4, pp. 2015–2024, 2023.
- [13] G. P. Mattia, A. Pietrabissa, and R. Beraldi, "A load balancing algorithm for equalising latency across fog or edge computing nodes," *IEEE Transactions on Services Computing*, vol. 16, no. 5, pp. 3129–3140, 2023.
- [14] M. H. Kashani and E. Mahdipour, "Load balancing algorithms in fog computing," *IEEE Transactions on Services Computing*, vol. 16, no. 2, pp. 1505–1521, 2022.
- [15] A. N. Alsheavi, A. Hawbani, X. Wang, W. Othman, L. Zhao, Z. Liu, S. H. Alsamhi, and M. A. Al-qaness, "Iot authentication protocols: Classification, trend and opportunities," *IEEE Transactions on Sustainable Computing*, 2024.
- [16] X. Wei and Y. Wang, "Popularity-based data placement with load balancing in edge computing," *IEEE Transactions on Cloud Computing*, vol. 11, no. 1, pp. 397–411, 2021.
- [17] A. Kishor, R. Niyogi, A. T. Chronopoulos, and A. Y. Zomaya, "Latency and energy-aware load balancing in cloud data centers: a bargaining game based approach," *IEEE Transactions on Cloud Computing*, vol. 11, no. 1, pp. 927–941, 2021.
- [18] S. Sefati, M. Mousavinasab, and R. Zareh Farkhady, "Load balancing in cloud computing environment using the grey wolf optimization algorithm based on the reliability: performance evaluation," *The Journal of Supercomputing*, vol. 78, no. 1, pp. 18–42, 2022.
- [19] S. Janakiraman and M. D. Priya, "Hybrid grey wolf and improved particle swarm optimization with adaptive inertial weight-based multi-dimensional learning strategy for load balancing in cloud environments," *Sustainable Computing: Informatics and Systems*, vol. 38, p. 100875, 2023.
- [20] A. Thakur and M. S. Goraya, "Raft: A hybrid metaheuristic based resource allocation framework for load balancing in cloud computing environment," *Simulation Modelling Practice and Theory*, vol. 116, p. 102485, 2022.
- [21] J. Luo, L. Rao, and X. Liu, "Spatio-temporal load balancing for energy cost optimization in distributed internet data centers," *IEEE Transactions on Cloud Computing*, vol. 3, no. 3, pp. 387–397, 2015.
- [22] M. I. Alghamdi, "Optimization of load balancing and task scheduling in cloud computing environments using artificial neural networks-based binary particle swarm optimization (bpps)," *Sustainability*, vol. 14, no. 19, p. 11982, 2022.
- [23] H. Gupta and K. Sahu, "Honey bee behavior based load balancing of tasks in cloud computing," *Int. J. Sci. Res.*, vol. 3, no. 6, pp. 842–846, 2014.
- [24] S. Sengupta, S. Basak, and R. A. Peters, "Particle swarm optimization: A survey of historical and recent developments with hybridization perspectives," *Machine Learning and Knowledge Extraction*, vol. 1, no. 1, pp. 157–191, 2018.
- [25] K. Yang, X. You, S. Liu, and H. Pan, "A novel ant colony optimization based on game for traveling salesman problem," *Applied Intelligence*, vol. 50, pp. 4529–4542, 2020.
- [26] S. Afzal and G. Kavitha, "Load balancing in cloud computing—a hierarchical taxonomical classification," *Journal of Cloud Computing*, vol. 8, no. 1, p. 22, 2019.
- [27] M. Noshay, A. Ibrahim, and H. A. Ali, "Optimization of live virtual machine migration in cloud computing: A survey and future directions," *Journal of Network and Computer Applications*, vol. 110, pp. 1–10, 2018.