symmetry

MDPI

Article

A Novel Hybrid MSA-CSA Algorithm for Cloud Computing Task Scheduling Problems

Shtwai Alsubai [1], Harish Garg [2,*] and Abdullah Alqahtani [3]

[1] Department of Computer Science, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Al-Kharj 11942, Saudi Arabia; sa.alsubai@psau.edu.sa
[2] School of Mathematics, Thapar Institute of Engineering & Technology (Deemed University), Patiala 147004, Punjab, India
[3] Department of Software Engineering, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Al-Kharj 11942, Saudi Arabia; aq.alqahtani@psau.edu.sa
* Correspondence: harishg58iitr@gmail.com

Abstract: Recently, the dynamic distribution of resources and task scheduling has played a critical role in cloud computing to achieve maximum storage and performance. The allocation of computational tasks in the cloud is a complicated process that can be affected by some factors, such as available network bandwidth, makespan, and cost considerations. However, these allocations are always non-symmetric. Therefore, it is crucial to optimize available bandwidth for efficient cloud computing task scheduling. In this research, a novel swarm-based task scheduling with a security approach is proposed to optimize the distribution of tasks using available resources and encode cloud information during task scheduling. It can combine the Moth Swarm Algorithm (MSA) with the Chameleon Swarm Algorithm (CSA) for the task scheduling process and utilizes the Polymorphic Advanced Encryption Standard (P-AES) for information security of cloud scheduled tasks. The approach offers a new perspective for utilizing swarm intelligence algorithms to optimize cloud task scheduling. The integration of MSA and CSA with P-AES enables the approach to provide efficient and secure task scheduling by exploiting the strengths of used algorithms. The study evaluates the performance of the proposed approach in terms of the degree of imbalance, makespan, resource utilization, cost, average waiting time, response time, throughput, latency, execution time, speed, and bandwidth utilization. The simulation is carried out using a wide range of tasks from 1000 to 5000. The results show that the approach provides an innovative solution to the challenges of task scheduling in cloud environments and improves the performance of cloud services in terms of effectiveness and security measures.

Keywords: security; task scheduling; cloud computing; hybrid model; advanced encryption standard; moth swarm algorithm; Chameleon Swarm Algorithm

Citation: Alsubai, S.; Garg, H.; Alqahtani, A. A Novel Hybrid MSA-CSA Algorithm for Cloud Computing Task Scheduling Problems. *Symmetry* **2023**, *15*, 1931. https://doi.org/10.3390/sym15101931

Academic Editors: Yunfei Fang, Peng Wu and Sergei D. Odintsov

Received: 1 September 2023
Revised: 4 October 2023
Accepted: 13 October 2023
Published: 18 October 2023

1. Introduction

In the realm of Internet computing, cloud computing signifies a recent revolution that provides numerous exceptional advantages over distributed computing. Leveraging large-scale computing resources, this multifunctional and highly efficient computing system ensures the effectiveness of cloud services. The process of scheduling and distributing tasks that necessitate processing is a fundamental aspect of cloud computing and involves numerous computing resources [1–3], providing users with customized computing, information, and storage services. Efficient task scheduling is crucial for successful task execution within cloud environments. As a result, implementing effective optimizations to task scheduling mechanisms can enhance the performance of CC services [4–6].

The task scheduling process involves scheduling task computation and resource allocation under specific constraints. The allocation of cloud computing resources for task computation mapping is determined based on specific optimization objectives in

this process [7,8]. Task scheduling considers the physical properties of resources and the attributes of task execution, ensuring the timely completion of tasks within a cost-effective framework for optimizing cloud computing utilization and preserving QoS requirements; it is imperative to choose a suitable task scheduling algorithm. Virtual machines are the primary computing elements utilized for task processing. Each virtual machine possesses distinct attributes, whether they are heterogeneous or homogeneous [9,10]. By utilizing the scheduling algorithm, the appropriate virtual machine for the task is chosen. Researchers focus on reducing makespan value and task execution time when selecting scheduling algorithms [11,12].

To process particular data points, the physical machine (PM) is required as per the VM plan. Static load balancing algorithms are utilized when the cloud environment experiences changes in workload. However, static algorithms are insufficient due to the dynamic nature of workload changes over time in cloud computing environments. Accordingly, it is crucial to implement dynamic schemes for distributing the workload among VMs [13,14]. Task scheduling is an NP-hard problem in the CC due to its dynamic and heterogeneous nature. While dynamic scheduling schemes enhance efficiency, they also increase communication overhead and difficulty, creating challenges for service providers [15,16].

Task scheduling in CC has been addressed using various techniques, but these algorithms are unsuitable for large-scale scheduling problems. Metaheuristic algorithms, such as ACO, PSO, GA, cuckoo search, whale optimization algorithm, symbiotic organisms search, and Harris Hawks optimization, have gained increased attention in recent years due to their efficiency in discovering optimal solutions in polynomial time [17,18]. However, some of these algorithms often suffer from premature convergence, making it difficult to overcome local minima in large solution spaces, resulting in less-than-optimal solutions that impact system performance and undermine QoS guarantees. Furthermore, the nonlinear models usually exhibit symmetry, non-convexity, and multiple solutions. Thus, new and flexible algorithms are required to efficiently compute the best global solution for task scheduling in CC environments [19,20].

In recent years, optimizing bandwidth in task scheduling has become an increasingly important problem. In cloud computing environments, virtualized resources are shared among multiple users and applications, making the efficient use of network bandwidth critical to ensure task performance and reliability, as well as effective use of cloud resources [21,22]. To balance competing demands of network bandwidth and task execution, various factors need to be considered, including data size, processing requirements, security needs, and network resource availability. Various techniques are employed for bandwidth optimization in task scheduling, such as fixed allocation, dynamic allocation, priority-based allocation, and Quality of Service (QoS) allocation. The selection of an approach is dependent on the specific requirements and constraints of the cloud environment, tasks being executed, and network conditions. However, optimizing bandwidth is a complicated issue that requires a multi-disciplinary approach, integrating network optimization, task scheduling, and security considerations. Therefore, further research is necessary to effectively integrate bandwidth optimization with task scheduling, aiming to strike a balance between network bandwidth and task execution trade-offs [23–25].

As cloud computing becomes increasingly prevalent, there is a growing necessity for assigning tasks to available resources in a manner that maximizes the overall efficiency of the cloud infrastructure. The optimization of task scheduling in large-scale systems using an MSA and CSA is driven by the objective of enhancing task scheduling efficiency, security, and reliability. A swarm intelligence-based optimization algorithm, MSA, is derived from the behavior of moths attracted to a light source. It has been employed to tackle optimization issues in several fields, including work schedules. However, MSA can suffer from premature convergence and slow convergence rates, which limit its effectiveness in complex optimization problems. Inspired by the adaptation of chameleons to their environments, CSA is a novel algorithm that utilizes swarm intelligence. CSA is more effective than other swarm intelligence algorithms in solving optimization problems, par-

ticularly those with high dimensionality and complexity. In the task scheduling problem, the combination of MSA and CSA strengths within the hybrid MSA-CSA algorithm leads to superior outcomes concerning the convergence speed. Additionally, the optimization of bandwidth and security in the scheduling process ensures that the resources are used effectively and that the system is protected from potential security threats. Summarizing the key highlights of this paper:

- A novel adaptive approach for optimal task transfer in CC is proposed to minimize the transfer of task time across available resources.
- A hybrid model-based scheduling framework is designed to achieve efficient task scheduling in CC while ensuring data security.
- This work proposes a novel scheduling model to enhance the efficiency of task scheduling while ensuring data security in the CC model. The framework is based on hybrid MSA-CSA models.
- The scheduling of tasks in the cloud incorporates P-AES to ensure data security.
- A scheduling algorithm that uses a hybrid meta-heuristic technique with low complexity has been developed and shows significant improvements in makespan, cost, degree of imbalance, resource utilization, average waiting time, response time, throughput, latency, execution time, speed, and bandwidth utilization.

In this research paper, the subsequent sections are arranged as follows: Section 2 presents a summary of the related studies concerning task scheduling in CC. In Section 3, the problem statement and formulation are presented, while Section 4 introduces the proposed approach for task scheduling optimization. This approach utilizes a hybrid MSA-CSA algorithm and incorporates the P-AES security method. The proposed approach's simulation results and performance evaluation are discussed in Section 5. In conclusion, Section 6 provides a summary of the paper and outlines potential avenues for future research.

2. Literature Review

Efficient task scheduling is a significant challenge in the realm of CC since it greatly affects system performance. Improving the task scheduling process is necessary by developing an efficient algorithm. In the following section, we delve into some of the most recent task-scheduling methods that are currently available.

Nabi et al. [26] presented a task scheduling approach that prioritizes compute-intensive and independent cloud tasks, aiming to reduce execution time and enhance throughput and cloud resource utilization. They introduced a new strategy for inertia weight named Leaner Descending and Adaptive Inertia Weight (LDAIW) for PSO-based algorithms, which enhances PSO model performance concerning throughput, ARUR, and makespan. By integrating global and local search more effectively, the LDAIW strategy offers improved performance. However, the presented approach may require substantial computational resources, limiting its feasibility in resource-constrained environments.

Zubair et al. [27] developed a modified scheduling method based on symbiotic organisms search (SOS) for mapping heterogeneous tasks onto cloud resources with varying capacities. The new approach streamlines the mutualism process of the algorithm by using equity as an indicator of the species' relationship effectiveness in the ecosystem, enabling them to progress to the subsequent generation. The approach utilizes a geometric mean instead of the initial mutual vector to improve the persistence benefits of two separate species. The modified algorithm, called G_SOS, aims to reduce various performance metrics while improving the convergence speed for optimal solutions in Infrastructure as a Service (IaaS) cloud environments. However, the presented approach may not be suitable for scenarios with homogeneous tasks or resources with similar capacities, as it is specifically designed to facilitate the effective allocation of heterogeneous tasks to utilize cloud resources with varying capacities.

Gupta et al. [28] presented modifications to the HEFT algorithm for rank generation and processor selection phases. These modifications enhance the accuracy of rank calcu-

lation and optimize the selection of available processor slots for task scheduling. These enhanced versions of the HEFT algorithm were presented to reduce the makespan of a given task submission on VMs while adhering to user-specified financial constraints. However, the presented model may increase the complexity of the algorithm and require significant computational resources for implementation, which may not be feasible for use in resource-constrained environments or for large-scale workflows.

Amer et al. [29] introduced ELHHO, a modified version of the HHO, to tackle the problem of multi-objective scheduling. To improve the exploration phase of the HHO algorithm, they utilized the elite opposition-based learning method. To avoid local optima and meet QOS requirements, the minimum completion time algorithm is utilized as the initial phase for each running time. This replaces the use of a random solution for achieving a determined initial solution. Maximizing resource utilization and minimizing the schedule length and execution cost are the primary goals. However, the presented approach is designed specifically for multi-objective scheduling problems, and it may not be optimal for single-objective problems or scenarios where other metrics, such as response time or throughput, are more crucial.

A research investigation was performed by Alboaneen et al. [30] to address the co-optimization problem of VM placement and task scheduling. MOA was utilized in this work for scheduling independent tasks for VMs and placing VMs on physical hosts (PHs). The study measured various parameters such as resource utilization, DOI, makespan, and execution cost. Even though the integration co-optimization approach may lead to better results, it could also increase the computational complexity of the problem, limiting its feasibility for use in resource-constrained environments. Albert et al. [31] present a cloud task scheduling solution using a hybrid Whale Harmony optimization algorithm. This study aims to accomplish load balancing in the system by minimizing the cost and makespan of a given task set. A small control parameter is used to stimulate the implementation of the WOA, which is inspired by the bubble-net feeding method. To ensure a high convergence rate, the fitness function is formulated by combining load imbalance, cost, energy consumption, resource utilization, and computation time. However, the use of a hybrid Whale Harmony optimization algorithm may increase the algorithm's complexity and result in longer processing times.

Alsadie et al. [32] presented the metaheuristic framework called MDVMA utilizing the NSGA-II algorithm to address the problem of dynamic VM allocation with optimal scheduling of tasks in the cloud. This framework enhances multiple objectives, such as makespan, energy usage, and cost, simultaneously and provides the Cloud Service Provider with a set of non-inferior solutions to choose from as per their requirements. However, using these algorithms for task scheduling may pose limitations, as it could require significant computational resources to find optimal or near-optimal solutions. Agarwal et al. [33] proposed a task scheduling method using PSO, integrating an opposition-based learning technique to prevent premature convergence and improve the speed of the PSO algorithm. The presented approach was compared with established task scheduling strategies such as mPSO, GA, minimum completion time, minimum execution time, and max-min. However, the PSO algorithm may encounter difficulties when dealing with larger and more intricate task sets, leading to potential inefficiencies. When the task count increases, the scalability of the PSO algorithm may become an issue, making it less suitable for large-scale task scheduling.

Wei et al. [34] developed an improved ACO to avoid local optimization in scheduling tasks. To optimize the overall performance, the algorithm utilizes reward and punishment coefficients for optimizing pheromone updating rules. Additionally, it uses dynamic updates of the volatility coefficient to enhance the optimization process. Additionally, the VM load weight coefficient is introduced to ensure load balance. However, the performance of the model may depend on the quality of the initial solution and parameter settings, which can be challenging to determine. Moreover, the effectiveness of the algorithm may be impacted by task diversity, resource requirements, and cloud environment complexity.

A task scheduling algorithm and conceptual model for dynamic resource allocation via task migration in VMs are presented by Ramasamy et al. [35]. Feature extraction is performed on the user's task demands, followed by feature reduction using the Modified PCA algorithm to decrease processing time. Hybrid PSO and Modified GA are utilized for resource allocation after combining both the user task and cloud server features. Finally, the task that has been optimized is scheduled to a specific VM to allocate the resources. The presented algorithm may not be flexible enough to handle dynamic changes in resource demands and server failures.

3. Problem Statement and Formulation

Cloud computing infrastructure comprises physical servers that are used for hosting VMs, which in turn are used for executing user tasks. The scheduling types in CC are the selection of server and task distribution. Optimizing task scheduling aims to balance various performance metrics, including bandwidth utilization, latency, make span, execution time, and cost. The objective function of optimization problems depends on the application and organization goals, such as maximizing data transfer, minimizing task completion time, makespan, or total service cost. Constraints include available bandwidth, storage, processing capacity, budget, data security, SLAs, and other performance metrics. Mathematical expressions for constraints and the objective function vary by the cloud computing system and organization goals. Task scheduling algorithms aim to allocate VMs according to user service requirements and VM status. However, existing algorithms often suffer from premature convergence, resulting in suboptimal solutions that undermine QoS guarantees. Therefore, there is a need for efficient and adaptable algorithms that can compute the most optimal global solution for scheduling tasks in CC environments.

Equation (1) depicts that the cloud system (CS) is composed of PMs, with each machine comprising VMs.

$$CS = \left[PM_1, PM_2, \ldots, PM_i, \ldots, PM_{Npm}\right] \tag{1}$$

The cloud's PM performance can be expressed as follows:

$$PM = [VM_1, VM_2, \ldots, VM_k, \ldots, VM_{Nvm}] \tag{2}$$

The *kth* VM is denoted as VM_k, where k ranges from 1 to N_{vm}. The number of VMs is denoted by Nvm, while VM_k represents the kth device of a virtual machine in the cloud.

The VM_k features is calculated as follows:

$$VM = [SIDV_k, mips_k] \tag{3}$$

$$T = [Task_1, Task_1, \ldots, Task_i, \ldots, Task_{tsk}] \tag{4}$$

$$Task_i = [SIDT_i, Task\text{-}length_i, ECT_i, LI_i] \tag{5}$$

The identity number of the ith task is denoted by $SIDT_i$, while the length of the task is denoted by $task\text{-}length_i$. The completion time for the ith task is represented by Time ECT_i, while the task preference in terms of the number of tasks N_{tsk} is denoted by LI_i.

The Expected Complete Time (ECT) metric is defined by Matrix (6) and has a size of $N_{tsk} \times N_{vm}$. This metric represents the amount of time needed to complete a task on each computing device, also known as a virtual machine (VM).

$$ECT = \begin{bmatrix} ECT_{1,1} & ECT_{1,2} & ECT_{1,3} & \cdots & ECT_{1,N_{vm}} \\ ECT_{2,1} & ECT_{2,2} & ECT_{2,3} & \cdots & ECT_{2,N_{vm}} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ ECT_{N_{tsk}}, 1 & ECT_{N_{tsk},}2 & \cdots & \cdots & ECT_{N_{tsk},N_{vm}} \end{bmatrix} \tag{6}$$

An effective algorithm is required to schedule tasks in CC systems, which can optimize several objectives while ensuring security. Based on the following considerations, the mathematical expression for the efficient task scheduling problem formulation is derived:

3.1. Decision Variables

ST_i: start time of task I.
ET_i: end time of task i.

3.2. Objective Function

3.2.1. Minimize the Maximum Execution Time

The main goal is to reduce the longest execution time of any task to a minimum. Therefore, the objective function is given by:

$$\text{minimize}(\max(\text{ET}_i)) \tag{7}$$

3.2.2. Maximize the Throughput

The aim is to optimize the task completion rate within a specified time frame by maximizing the number of tasks that can be finished. Therefore, the objective function is expressed as:

$$\text{maximize}\left(\frac{n}{\max(\text{ET}_i)}\right) \tag{8}$$

where the total number of tasks is 'n'.

3.2.3. Minimize the Average Execution Time

The main goal is to reduce the mean time required for all tasks to finish their execution. Therefore, the objective function is expressed as:

$$\text{minimize}\left(\frac{\text{sum}(\text{ET}_i)}{n}\right) \tag{9}$$

3.2.4. Maximize the Total Bandwidth Usage

The aim is to optimize the total data transfer between tasks to maximize it. Therefore, the objective function is given by:

$$\text{maximize}(\text{sum}(\text{BW}_i)) \tag{10}$$

where BW_i is the bandwidth requirement of task i.

3.2.5. Minimize the Total Cost

Aiming to reduce the overall cost of resources utilized in accomplishing the tasks. Therefore, the objective function is given by:

$$\text{minimize}(\text{sum}(\text{Cost})) \tag{11}$$

where Cost is the cost of resources used to complete the tasks.

3.2.6. Minimize the Total Execution Time

The main goal is to reduce the overall time needed to finish all the tasks. Therefore, the objective function is given by:

$$\text{minimize}(\text{sum}(\text{ET}_i)) \tag{12}$$

3.3. Constraints

3.3.1. Security Constraints

a Confidentiality constraint: All data must be encrypted during transmission and storage.
b Integrity constraint: Data must not be modified during transmission or storage.
c Availability constraint: The cloud system must be available for task scheduling at all times.

3.3.2. Resource Availability Constraints

- CPU constraints

The CPU constraint ensures that the total CPU usage does not exceed the available CPU capacity. The sum of CPU usage of all tasks, calculated as the product of the start and end time of each task and its CPU requirement, should be less than or equal to the total CPU capacity available.

$$\text{sum}(\text{ST}_i\text{CPU}_i) \leq \text{T}_{\text{cpu}} \tag{13}$$

- Memory constraint

The Memory constraint ensures that the total memory usage does not exceed the available memory capacity. The sum of memory usage of all tasks, calculated as the product of the start and end time of each task and its memory requirement, should be less than or equal to the total memory capacity available.

$$\text{sum}(\text{ST}_i Mem_i) \leq \text{T}_{\text{mem}} \tag{14}$$

- Storage constraint

The Storage constraint ensures that the total storage usage does not exceed the available storage capacity. The sum of storage usage of all tasks, calculated as the product of the start and end time of each task and its storage requirement, should be less than or equal to the total storage capacity available.

$$\text{sum}(\text{ST}_i \times \text{Storage}_i) \leq \text{T}_{\text{storage}} \tag{15}$$

- Network bandwidth constraint

The Network bandwidth constraint ensures that the total bandwidth usage does not exceed the available bandwidth capacity. The sum of the bandwidth requirement of all tasks, divided by the execution time of each task, should be less than or equal to the total bandwidth capacity available.

$$\text{sum}\left(\frac{\text{BW}_i}{\text{ET}_i}\right) \leq T_{bw} \tag{16}$$

3.3.3. Deadline Constraint

The Deadline constraint ensures that all tasks are completed before their respective deadlines. This constraint is formulated as ET_i being less than or equal to DD_i.

$$\text{ET}_i \leq \text{DD}_i \text{ for all i.} \tag{17}$$

3.3.4. Makespan Constraint

The Makespan constraint ensures that the maximum time taken by any task to complete execution does not exceed T_{max}. This constraint is formulated as $Max(ET_i)$ being less than or equal to T_{max}.

$$\text{Max}(\text{ET}_i) \leq \text{T}_{\text{max}} \tag{18}$$

3.3.5. Throughput Constraint

Ensuring that the number of tasks completed within a specific period is equal to or greater than a predefined value Tp is the purpose of the throughput constraint. This constraint is formulated as n divided by the maximum execution time of all tasks being greater than or equal to T_p.

$$\frac{n}{T_{\max}} \geq T_p \tag{19}$$

3.3.6. Latency Constraint

The constraint on Latency guarantees that the mean duration for all task completions is not greater than a specific value T_l. This constraint is formulated as the total execution times of all tasks divided by the number of tasks being less than or equal to T_l.

$$\frac{sum(ET_i)}{n} \leq T_l \tag{20}$$

3.3.7. Bandwidth Constraint

The Bandwidth constraint ensures that the total amount of data transferred between tasks is greater than or equal to a certain value T_b. This constraint is formulated as the sum of the bandwidth requirements of all tasks being greater than or equal to T_b.

$$\text{sum}(\text{BW}_i) \geq \text{T}_b \tag{21}$$

3.3.8. Cost Constraint

The Cost constraint ensures that the total cost of resources used to complete the tasks is less than or equal to a certain value T_c. This constraint is formulated as the total cost of resources used by all tasks being less than or equal to T_c.

$$\text{sum}(\text{Cost}) \leq \text{T}_c \tag{22}$$

where T_cpu, T_mem, T_storage, T_bw, T_max, T_p, T_l, T_b, and T_c are the available resources, maximum execution time, minimum throughput, maximum latency, minimum bandwidth, and maximum cost, respectively. All of the above constraints are represented as inequalities that must be satisfied by the variables.

In this approach, the weighted sum method is utilized, where each metric is given a weight, and the optimization goal is to minimize the sum of the weighted metrics. The objective function can then be represented as:

$$minimize\ w_1 Max(ET_i) + \frac{w_2 n}{T_{max}} + \frac{w_3 sum(ET_i)}{n} + w_4 sum(BW_i) + w_5 sum(Cost) + w_6 sum(ET_i) \tag{23}$$

where w_1, w_2, w_3, w_4, w_5, and w_6 are the weights assigned to each metric.

In the above objective function, the first term represents the makespan; the second term represents the throughput, the third term represents the latency, the fourth term represents the bandwidth, the fifth term represents the cost, and the sixth term represents the execution time. The optimization problem seeks to determine the start and end times that optimize each task's execution, meeting all constraints and minimizing the overall execution time. The optimization problem is solved using the Hybrid Moth Swarm and Chameleon Swarm algorithm.

4. Scheduling of Tasks in the Cloud

In this study, we focus on the optimized transfer of tasks to the cloud and optimal scheduling of the task to VMs with security using P-AES [36]. Cloud task scheduling aims to allocate computing resources efficiently to execute user tasks. Cloud data centers use several physical servers, each running multiple virtual machines, to provide computing services to various applications. Task scheduling allocates and executes user tasks on

VMs based on processing capability and computing resource cost. Multiple objectives can be taken into account to enhance task scheduling performance in the cloud data center, including resource utilization, execution time, degree of imbalance, bandwidth utilization, speed, latency, makespan, and throughput. In this regard, we propose a hybrid MSA and CSA to optimize the CC environment's task scheduling process. The algorithm can allocate virtual machines (VMs) to user-submitted tasks according to their needs while also achieving multiple objectives of the cloud data center. These objectives may include maximizing bandwidth utilization and throughput and minimizing makespan, execution time, bandwidth, and cost. Figure 1 shows the architecture of the proposed work.

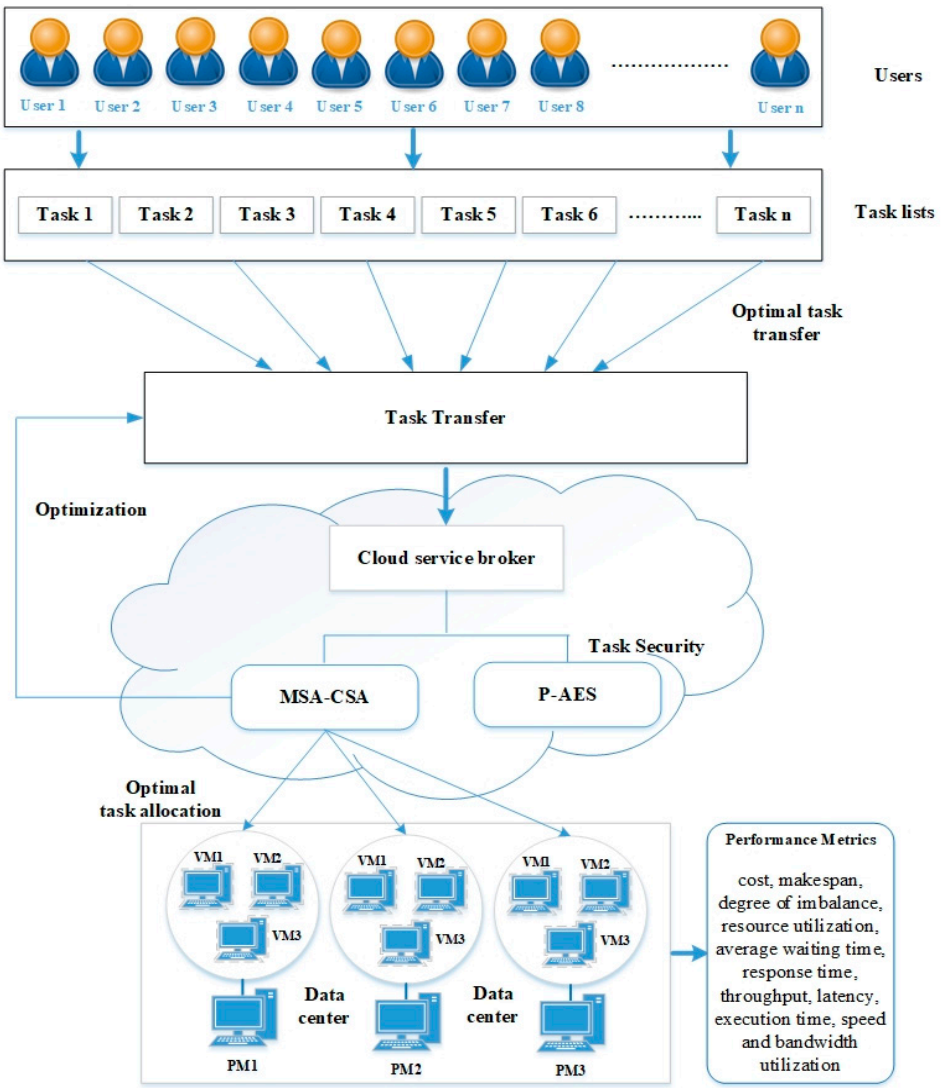

Figure 1. The architecture of the proposed methodology in task scheduling.

We implement this algorithm in Amazon Web Services (AWS) cloud, which is one of the leading cloud service providers. To ensure the security of the task data during transfer and processing, we use Polymorphic AES, which is a highly secure encryption algorithm. The proposed algorithm will take into account various factors, including bandwidth usage, latency, makespan, execution time, cost, and more, to enhance the scheduling process within the cloud environment. In the upcoming sections, we will elaborate on the proposed algorithm.

4.1. Security Strategy

Ensuring security is a critical aspect when dealing with cloud mechanisms and scheduling procedures. Attacks such as data tampering and information interception can occur during scheduling, making security a fundamental concern. To address this issue, the P-AES technique is employed to provide security to workflow scheduling. P-AES operates in 128 distinct ways, which makes it challenging for attackers to obtain the encryption key and decipher the cipher's precise structure. P-AES determines the operation specifics during runtime for communicating parties, employing a subset of the key bits.

Polymorphic Advances Encryption Standard (P-AES)

Initially, P-AES operates differently from the conventional AES by processing a single 16-byte block at a time without requiring additional padding for smaller input sizes. This approach minimizes data fragmentation and potential vulnerabilities associated with block-wise processing. Additionally, P-AES can accommodate key lengths of 16, 24, or 32 bytes, offering flexibility in key management and increasing the complexity of encryption, thereby enhancing overall security.

The P-AES encryption process consists of multiple stages, including the ModifiedSubBytes, ModifiedShiftRows, and ModifiedMixColumns stages. Each stage involves intricate data manipulations and circular bit shifts, making it significantly more challenging for potential attackers to decipher the encrypted information. Furthermore, polymorphism is employed in the ModifiedSubBytes stage, where each byte's bits are circularly moved to the left by a 7-byte substitution index. This polymorphic characteristic generates distinct ciphertexts for the same plaintext with different encryption keys, further augmenting the encryption's security.

Moreover, the P-AES algorithm employs a dynamic reordering of rows in the ModifiedMixColumns matrix during encryption based on the column_mixing_index integer. This dynamic modification makes the encryption process more resistant to cryptanalysis attempts, adding an additional layer of security. By incorporating P-AES into task scheduling, cloud computing environments can ensure that sensitive data remains confidential and safeguarded from unauthorized access during the allocation and processing of computational tasks. After implementing the security approach, the allocation of tasks in the cloud computing environment is performed. A novel swarm-based approach that combines the MSA and the CSA to achieve efficient task scheduling. Al-Attar Ali Mohamed proposed the moth swarm algorithm (MSA) in 2016 [37]. This algorithm is a new population-based optimization method that takes inspiration from how moths navigate towards moonlight in a noisy environment. Although the MSA excels in swarm intelligence, its convergence precision and speed may be restricted in certain applications. This paper aims to enhance the convergence speed and accuracy of the original algorithm by using the GSA.

4.2. MSA

The position of a light source in the MSA represents a potential solution to the optimization problem, and the brightness of the light source is regarded as the fitness of this solution. The proposed algorithm utilizes these assumptions to estimate the features. Furthermore, the moth swarm can be categorized into three distinct groups of moths, each with its own specific characteristics.

Pathfinders: There is a small group of moths that can find new areas in the optimization space. The primary goal of this type is to determine the optimal positions of the light source to direct the movement of the main swarm.

Prospectors: A collection of moths that typically travels in a haphazard spiral pattern near illuminated areas designated by the explorers. Onlookers: A collection of moths that move towards the optimal global solution discovered by prospectors. The pathfinder's positions and guidance for the next update iteration are regarded as the most effective fitness function in the swarm. Therefore, the second and third best categories are referred

to as prospectors and onlookers, correspondingly. The suggested optimization algorithm is carried out in the subsequent stages.

4.2.1. Pathfinder Phase

At the beginning of the MSA, the positions of moths are generated in a random manner. This process takes into account the problem's dimensions (D) and the size of the population (n):

$$y_{ij} = rand[0,1] \cdot \left(y_j^{max_j^{min}} + y_j^{min} \right) \tag{24}$$

Local optimization can sometimes cause the moths to decline. In order to prevent early convergence and enhance the variety within the population, a portion of the swarm has the ability to move away from the locally optimal solution. The pathfinder moths keep track of their location by engaging in crossover operations and possess the capacity to travel over great distances by utilizing the suggested adaptive crossover with Lévy mutation. This method can be explained as follows:

4.2.2. Choice of Crossover Points

One potential approach to address diversity is by choosing the positions for crossover. The measurement of the normalized dispersal degree σ_j^t of the individuals in the jth dimension is conducted at iteration t.

$$\sigma_j^t = \frac{\sqrt{\frac{1}{n_p}\left(\sum_{ij}^t y_{ij}^t - y_j^t\right)^2}}{y_j^t} \tag{25}$$

$$\mu^t = \frac{1}{d}\sum_{j=1}^{d} \sigma_j^t \tag{26}$$

The c_p (crossover point group) is defined as

$$j \in c_p \; if \; \sigma_j^t \leq \mu^t \tag{27}$$

4.2.3. Lévy Mutation

Lévy motions are random processes that use α-stable distribution and can cover large distances with varying step sizes. The Lévy α-stable distribution is closely connected to heavy-tailed probability density functions, fractal statistics, and anomalous diffusion. Lévy distribution has a heavier tail compared to Gaussian and Cauchy distributions. Mantegna's algorithm generates random samples Li that mimic the behavior of Lévy flights, allowing for the emulation of α-stable distribution.

$$L_i \sim step \oplus Levy(\alpha) \sim 0.001 \frac{\mu}{\left|x^{1/\alpha}\right|} \tag{28}$$

4.2.4. Position Update

The sub-trial vector equation is as follows:

$$\vec{v_p} = \vec{y}_{r^1}^t + L_{p1}^t \cdot \left(y - \vec{y_{r^3}^t} \right) + L_{p^2}^t \cdot \left(\vec{y_{r^4}^t} - \vec{y_{r^5}^t} \right) \tag{29}$$

The position of each pathfinder solution (host vector) is updated by integrating the mutated d variables. The v_{pj}^t trail solution is described as

$$v_{pj}^t = \begin{cases} v_{pj}^t & if \quad j \in c_p \\ y_{pj}^t & if \quad j \in c_p \end{cases} \tag{30}$$

Once the previous procedure is completed and compared to the host solution, the fitness value of the sub-trail solution is computed. The next generation is determined by selecting the most suitable solutions, which are described for the purpose of minimizing the problem as follows:

$$\vec{y_p^{t+1}} = \begin{cases} \vec{y_p^t} & if \quad f\left(v_p^t\right) \geq f\left(\vec{y_p^t}\right) \\ \vec{v_p^t} & if \quad f\left(v_p^t\right) \geq f\left(\vec{y_p^t}\right) \end{cases} \tag{31}$$

The estimated probability value p_e is directly related to the luminescence intensity fit_p.

$$p_e = \frac{fit_p}{\sum_{p=1}^{n_p} fit_p} \tag{32}$$

The calculation of luminescence intensity fit p involves determining the objective function value f_p in order to solve minimization problems.

$$fit_p = \begin{cases} \frac{1}{1+f_p} & for \quad f_p \geq 0 \\ 1 + \left|f_p\right| & for \quad f_p < 0 \end{cases} \tag{33}$$

4.2.5. Prospector Phase

The prospector moths fly in a logarithmic spiral path due to their transverse orientation. The group of prospector moths is regarded as the next finest in terms of luminescence intensity. As the iterations T progress, the number of prospectors, denoted as n_e, is processed in a way that gradually decreases.

$$n_e = round\left((n - n_c) \times \left(1 - \frac{t}{T}\right)\right) \tag{34}$$

Once the pathfinders finish searching, they share the brightness levels with the prospectors. The prospectors then use this information to update their locations and find new sources of light. Every prospector searches extensively around their assigned light source in a spiral pattern. The artificial light source x_e is chosen based on the positions of the pathfinder moths, using probability p_e as determined by Equation (35). The location of the ith prospector moth is stated in the following manner.

$$y_i^{t+!} = \left[y_i^t - y_p^t\right] \times \ e^{\theta} \times cos2\pi\theta + y_p^t \tag{35}$$

When a prospector moth discovers a solution that is brighter than current light sources, it transforms into a pathfinder moth. Certainly, the new light sources combined at the end of this phase.

4.2.6. Onlooker Phase

Onlookers in a moth swarm are the ones with the least bright luminescence. Reducing the number of prospectors in the optimization process increases the number of onlookers $(n_u = n - n_e - n_c)$. This, in turn, speeds up the convergence of the algorithm towards a global solution. These moths are attracted to the brightest light source (the moon) and fly towards it. The MSA aims to improve search efficiency by focusing on the most promising areas for prospectors. The observers are split into two categories.

4.2.7. Gaussian Walks

Initially, The number of onlookers is equal to half of n_u, rounded to the nearest whole number. The new observers move based on Gaussian distributions using Equation (36). The moths in subgroup $x_i^{t+!}$ move in a pattern called Gaussian walks, which is described by Equation (37).

Gaussian distribution is

$$f(p) = \frac{1}{\sqrt{2\pi}\sigma_G} exp\left(-\frac{(p-\mu)^2}{2\sigma_g^2}\right) \quad -\infty < p < \infty \quad p \sim N(\mu,) \tag{36}$$

$$y_i^{t+1} = y_i^t + \varepsilon_1 + \left\lfloor \varepsilon_2 \times gbest^t - \varepsilon_3 \times y_i^t \right\rfloor \quad \forall i \in \{1, 2, \ldots, n_o\} \tag{37}$$

$$\varepsilon_1 \sim random(size(d)) \oplus N(best_g^t, \frac{logt}{t}(y_i^t - best_g^t)) \tag{38}$$

4.2.8. Associative Learning Mechanism with Immediate Memory

In the second part, the number of onlookers is equal to the difference between the total number of users and the number of observers. Moth behavior is greatly affected by associative learning and short-term memory, lasting only a few seconds. Moths rely on associative learning for communication. The second group of onlookers is programmed to move towards moonlight based on associative learning and can imitate the behavior of moths. The immediate memory is set using a uniform Gaussian distribution between $y_i^t - y_i^{min_i^{max_i^t}}$. The equation used to update this type is

$$y_i^{t+1} = y_i^t + 0.001.G\left\lfloor y_i^t - y_i^{min_i^{max_i^t}} + (1 - g/G)\cdot r_1(best_p^t - y_i^t) + \left|2g/G \cdot r_2 \cdot (best_g^t - y_i^t)\right|\right\rfloor \tag{39}$$

After every round, the fitness value of the population is utilized to determine the type of every moth for the following iteration.

4.3. CSA

In 2021, Braik [38] introduced the CSA, which imitates the food-searching and hunting mechanism of chameleons. The species that possess the ability to alter their color and blend seamlessly with their environment are highly specialized and capable of thriving in diverse environments.

4.3.1. Evaluation of Initialization and Function

The search process in CSA is initiated by generating a population of individuals at random, as it is a population-based meta-heuristic algorithm. The search area is d-dimensional, and a chameleon population of size n is generated, with each individual representing a possible solution to the optimization problem. Equation (40) characterizes the position of each chameleon in the search area at any given iteration.

$$y_t^i = \left[y_{t,1}^i, y_{t,2}^i, ..., y_{t,d}^i\right] \tag{40}$$

Equation (41) shows that the generation of the initial population is determined by the problem dimension and the number of chameleons in the search space:

$$y^i = l_j + r(u_j - l_j) \tag{41}$$

The evaluation of the fitness function determines the solution quality of every new position in every step.

4.3.2. Searching for a Target

The updating strategy of the chameleons' positions during the search is characterized by Equation (42):

$$y_{t+1}^{i,j} = \begin{cases} y_t^{i,j} + P_1\left(P_t^{i,j} - G_t^j\right)r_2 + P_2\left(G_t^j - y_t^{i,j}\right)r_1 \\ y_t^{i,j} + \mu\left(u^j - l^j\right)r_3 + l_b^j sgn(rand - 0.5)\end{cases} r_1 \geq P_p \quad r_1 < P_p \quad . \tag{42}$$

4.3.3. Eyes Rotation of Chameleon

Chameleons can spot their prey in a complete 360 degrees by utilizing their eye-rotating ability to identify its position. The following steps occur in the following sequence:

- The initial location or starting point of the chameleon is the center of gravity or focal point.
- The location of the prey can be identified by computing the rotation matrix.
- The location of the chameleon at the focal point is updated using the rotation matrix.
- Finally, they are brought back to their initial position.

4.3.4. Hunt of Target

Chameleons initiate an attack on their target when it approaches them closely. It has the closest distance to the prey, which is considered to be the optimal solution. The prey is attacked by the chameleon using its tongue. The chameleon's position can be enhanced as it is capable of extending its tongue up to twice its length. As a result, the chameleon can efficiently use its hunting space and successfully capture its prey. Notations and descriptions are shown in Table 1. Equation (43) can be used to mathematically model the velocity of a chameleon's tongue as it extends toward its prey:

$$v_{t+1}^{i,j} = wv_t^{i,j} + c_1\left(G_t^j - y_t^{i,j}\right) + c_2\left(P_t^{i,j} - y_t^{i,j}\right) + c_2\left(P_t^{i,j} - y_t^{i,j}\right)r_2 \quad (43)$$

Table 1. Notations and descriptions.

Notations	Descriptions
N_{pm}	Number of physical machines
N_{vm}	Number of virtual machines
VM_k	kth VM device
$mips_k$	processing acceleration of VMs by millions-of-instructions-per second
N_{tsk}	Number of tasks
$Task_i$	ith task
$SIDT_i$	ith task identity number
$length_i$	task length
ECT_i	ith task execution time
LI_i	task preference
$ETC_{i,j}$	ECT for the lth task on the jth VM
d	dimensions
n	number of moths
Di	distance between the ith moth and the jth flame
l	current repetition number
T	Total number of flames
N	Maximum number of flames
$y_{t,d}^i$	chameleon's position
t and $(t + 1)$	iteration step
$y_t^{i,j}$	current position
$y_{t+1}^{i,j}$	new position
$P_t^{i,j}$	best position
G_t^j	global best position
$v_{t+1}^{i,j}$	ith chameleon's new velocity
$v_t^{i,j}$	ith chameleon's current velocity

4.4. Optimized Task Scheduling Using Hybrid MSA-CSA

Cloud computing has become a widely adopted paradigm for delivering on-demand computing resources over the internet. However, efficient task scheduling remains a crucial challenge in maximizing the performance of cloud computing environments. To address this issue, our research proposes a novel hybrid algorithm that leverages two powerful optimization techniques: MSA and CSA.

The algorithm initiates the task allocation phase by employing MSA. During this phase, a population of potential solutions, represented as moths, is initialized to determine optimal task assignments to virtual machines (VMs). The fitness of each solution is evaluated based on important performance metrics, including makespan, throughput, and latency. Through a process of exploration and exploitation, the moths converge towards promising regions in the search space, representing optimized task assignments to VMs. The iterative refinement process of MSA ensures continuous improvement in the task allocation to VMs, striving for enhanced performance.

Once the task allocation phase is completed, the focus shifts to VM-to-PM allocation using CSA. Similar to MSA, CSA initializes a population of potential solutions, represented as chameleons, to determine the optimal allocation of VMs to PMs. The fitness of each solution is assessed based on resource utilization and cost considerations. The adaptive nature of CSA allows chameleons to adjust their "colors" (representing VM assignments) to match the available resources of PMs. This dynamic adaptation ensures that VMs are optimally distributed across PMs, leading to efficient resource allocation.

The hybrid algorithm integrates MSA and CSA in an iterative manner. MSA optimizes the initial task assignment to VMs, establishing a solid foundation for a well-distributed task schedule. Then, CSA refines the VM-to-PM allocation, ensuring effective resource utilization throughout the cloud infrastructure. The interplay between these two algorithms facilitates comprehensive optimization of the overall task-scheduling process.

The iterative optimization process continues until convergence, where an optimal solution for task scheduling in the cloud environment is achieved. By synergistically combining the strengths of MSA and CSA, our proposed hybrid approach effectively addresses the challenges of task assignment and resource allocation in cloud computing. The hybrid algorithm's adaptability to varying cloud computing demands, along with its ability to balance critical performance metrics such as makespan, latency, throughput, resource utilization, and cost efficiency, makes it a powerful tool for optimizing task scheduling in cloud computing environments.

In Table 2, each row denotes a task that needs to be scheduled to a VM that has the required resources to perform the task. The required resources for each task are listed in the "Required Resources" column. The Moth Swarm Algorithm and Chameleon Swarm Algorithm are used to determine which VMs are best suited to perform each task, and the results are recorded in the "Moth Swarm Algorithm" and "Chameleon Swarm Algorithm" columns, respectively. Finally, the "Assigned Resource" column lists the VM that has been assigned to perform each task based on the results of both algorithms. The pseudo-code for hybrid MSA-CSA is shown in Algorithm 1.

Table 2. Task Scheduling using Hybrid MSA-CSA on VMs.

Task	Required Resources	Moth Swarm Algorithm	Chameleon Swarm Algorithm	Assigned Resource
T1	CPU, 2 GB RAM	V1, V2	V2, V3	V2
T2	GPU, 4 GB RAM	V2, V3	V3, V4	V3
T3	CPU, 1 GB RAM	V3, V4	V4, V5	V4
T4	GPU, 2 GB RAM	V1, V4	V4, V5	V4
T5	CPU, 2 GB RAM	V2, V5	V5, V1	V5

Algorithm 1: Pseudo code for MSA-CSA

```
function hybridMSACSA(taskList, VMList, PMList):
// Initialize MSA parameters
MSA_maxIterations = 100
MSA_populationSize = 50
MSA_c1 = 1.5
MSA_c2 = 1.5
MSA_w = 0.8
// Initialize CSA parameters
CSA_maxGenerations = 50
CSA_populationSize = 30
CSA_mutationRate = 0.01
// Initialize hybrid algorithm parameters
hybrid_iterations = 10
hybrid_populationSize = 20
// Initialize global best solution
globalBestSolution = null
globalBestFitness = INF
// Run hybrid algorithm for a fixed number of iterations
for i = 1 to hybrid_iterations:
// Run MSA to optimize task assignment to VMs
MSA_solutions = initializeMSA(MSA_populationSize)
MSA_globalBestSolution = null
MSA_globalBestFitness = INF
for j = 1 to MSA_maxIterations:
for each solution in MSA_solutions:
fitness = evaluateFitness(solution, taskList, VMList)
if fitness < MSA_globalBestFitness:
MSA_globalBestSolution = solution
MSA_globalBestFitness = fitness

updateMSAPositions(MSA_solutions, MSA_globalBestSolution, MSA_c1, MSA_c2, MSA_w)
// Run CSA to optimize allocation of VMs to PMs
CSA_population = initializeCSA(CSA_populationSize)
CSA_globalBestSolution = null
CSA_globalBestFitness = INF
for k = 1 to CSA_maxGenerations:
for each chameleon in CSA_population:
fitness = evaluateFitness(chameleon, VMList, PMList)
if fitness < CSA_globalBestFitness:
CSA_globalBestSolution = chameleon
CSA_globalBestFitness = fitness

mutateCSA(CSA_population, CSA_globalBestSolution, CSA_mutationRate)

// Combine MSA and CSA solutions to create hybrid solution
hybridSolution = combineSolutions(MSA_globalBestSolution, CSA_globalBestSolution)
hybridFitness = evaluateHybridFitness(hybridSolution, taskList, VMList, PMList)
// Update global best solution for the hybrid algorithm
if hybridFitness < globalBestFitness:
globalBestSolution = hybridSolution
globalBestFitness = hybridFitness
return globalBestSolution
```

In conclusion, MSA is used to find a good initial solution for CSA. CSA then uses this solution as a starting point to explore the search space and find better solutions. This hybrid approach has been shown to be effective for solving optimization issues in various

scenarios like bandwidth, latency, throughput, makespan, execution time, and cost [39–41]. The objective function's mathematical expression is defined as:

$$\text{minimize}(\max(\text{ET}_i)) \tag{44}$$

$$\text{Maximize}\left(\frac{n}{\max(\text{ET}_i)}\right) \tag{45}$$

$$\text{Minimize}\left(\frac{\text{Sum}(\text{ET}_i)}{n}\right)$$

$$\text{maximize}(\text{sum}(\text{BW}_i)) \tag{46}$$

$$\text{minimize}(\text{sum}(\text{Cost})) \tag{47}$$

$$\text{minimize}(\text{sum}(\text{ET}_i)) \tag{48}$$

5. Experimental Results and Analysis

This section outlines the experimental configurations for the novel hybrid algorithms that schedule the transfer of cloud tasks and allocate them to VMs. The results of the system based on the experiments are discussed. This section also features a comparison of the outcomes of the proposed algorithm with those of prior research on cloud task scheduling.

Combining the MSA-CSA approach, as described in Section 4, is a novel method developed in this paper to optimize the scheduling of cloud task transfer and VMs. The primary goal of the algorithm proposed is to minimize the time taken for transferring cloud tasks and ensure the security of the transfer process using the implementation of a P-AES algorithm. The proposed hybrid swarm algorithm first utilizes the MSA to distribute the transfer tasks among available VMs based on their workload, which is determined by factors such as transfer speed, CPU, throughput, and machine storage. This step ensures that the transfer tasks are allocated in an efficient and balanced manner. Next, the CSA is applied to optimize the task schedule by transferring some tasks through other VMs to reduce the transfer time. This is achieved via position and speed that enhance the initial population of the algorithm generated by MSA. To ensure the security of the cloud task transfer process, a polymorphic AES algorithm is used to encrypt the data in transit and prevent unauthorized access to the information. In summary, the proposed hybrid swarm algorithm that combines MSA and CSA, along with the use of a polymorphic AES algorithm, provides an efficient and secure solution for cloud task transfer and scheduling on virtual machines. The algorithm optimizes the workload of VMs, enhances the transfer process by finding better possibilities for task transfer, and secures the data in transit to prevent any unauthorized access to the information.

5.1. Experimental Environment

This study employed a simulation of the AWS cloud environment to conduct the experiments. This paper presents a summary of the simulation parameters used in the experiments, which are displayed in Table 3. A Python toolbox is employed to facilitate efficient simulation. The python toolbox used in this study is highly useful for simulating distributed networks, including hybrid cloud environments. The experimental hardware setup consisted of a notebook computer with an Intel(R) Core(TM) i5-7300HQ CPU @2.8 GHz, a 64-bit Windows 10 operating system, and 4 GB RAM.

Table 3. Simulation Setting.

Entity	Parameter	Values of Settings
Hosts	Bandwidth	2 Gb/s
	Storage	500 GB
	RAM	1 GB
	No. of hosts	1
Virtual Machine	Bandwidth	2 Gb/s
	Size	20,000
	MIPS	100–1000
	No. of CPU	1
	Operation system	Windows
	RAM	2 GB
Datacenter	No. of data center	1
Cloudlets	Number of cloudlets	1000–5000
	Length	1000–2000

5.2. Parameter Setting

Table 4 column shows the parameters and values of two swarm intelligence-based algorithms, the Moth Swarm Algorithm and the Chameleon Swarm Algorithm, used for scheduling tasks in CC. MSA uses a swarm size of 50, runs for 50 iterations, and has a light absorption value of 0.5, a step size of 0.1, and an attraction exponent of 1.0. On the other hand, the Chameleon Swarm Algorithm also has a swarm size of 50 and runs for 50 iterations but uses a crossover probability of 0.8, a mutation probability of 0.1, a mutation rate of 0.1, and a local search method performed on 10% of the population. These algorithms aim to optimize the allocation process, and the parameters play a critical role in determining their performance. The population size parameter determines the number of individuals in the population, and the maximum number of iterations specifies the number of times the algorithm can update the population. The step size parameter controls the distance individuals move in the search space, while the attraction exponent sets the strength of attraction between individuals. The light absorption parameter controls the decay rate of the attraction force. In the Chameleon Swarm Algorithm, the crossover probability determines the likelihood of genetic material exchange, while the mutation probability and rate specify the probability of a gene modification. Finally, the local search method iteratively improves the fitness of a population subset. It is essential to select and tune the parameters to achieve efficient and effective task scheduling, and testing various parameter combinations can determine optimal values for specific tasks and workloads.

Table 4. Values considered for different parameters.

Algorithm	Parameter Name	Parameter Value
Moth Swarm	Swarm size	50
	Number of iterations	50
	Light absorption	0.5
	Step size	0.1
	Attraction exponent	1
Chameleon Swarm Algorithm	Swarm size	50
	Number of iterations	50
	Mutation rate	0.1
	Crossover probability	0.8
	Mutation probability	0.1
	Local search	10%

5.3. Evaluation Parameters

Applying appropriate evaluation parameters is necessary for achieving accurate assessment. Table 5 shows the measure of various parameters for scheduling tasks in the cloud.

Table 5. Various evaluation parameters.

Parameter	Description
Makespan	The time is taken to complete all tasks in the cloud environment
Throughput	The amount of work completed per unit of time in the cloud environment
Latency	The time is taken for data to travel from source to destination in the cloud environment
Bandwidth	The amount of data that can be transferred in a unit of time in the cloud environment
Cost	The total cost incurred in the cloud environment
Execution time	The total time taken for all tasks to complete in the cloud environment
Degree of imbalance	The difference between the highest and lowest loads across all nodes in the system
Resource utilization	The proportion of available resources that are being used by the system
Average waiting time	The average time that a request spends in the queue before being serviced by a node
Response time	The time it takes for a request to be processed by a node and receive a response
Speed	The rate at which a node can process requests
Bandwidth utilization	The proportion of available network bandwidth that is being used by the system

5.4. Discussion on the Comparison

This study compares various algorithms for task transfer and scheduling in CC using different parameters for different numbers of tasks (1000–5000). The proposed algorithm is compared with four other algorithms: HESGA, G_SOS, ANN-BPSO, and MALO. HESGA is a hybrid meta-heuristic algorithm that combines two different algorithms, ESA and GA. ESA optimizes task scheduling by considering task requirements and resource availability, while GA searches for the optimal solution by generating a population of candidate solutions and iteratively improving them. G_SOS is a population-based algorithm that simulates symbiotic relationships in biological ecosystems to find the optimal solution. The modified version, G_SOS, introduces several improvements over the original SOS algorithm. ANN-BPSO is a hybrid algorithm that combines ANN, a machine learning algorithm designed to learn from data, and BPSO, a population-based meta-heuristic algorithm. The ANN-BPSO model utilizes a neural network to predict the resource requirements of each task and then uses BPSO to allocate resources. MALO is a meta-heuristic algorithm based on the behavior of ant lions and optimizes task scheduling by considering task requirements and resource availability.

5.4.1. Makespan Result

Makespan is defined as the total length of a schedule required to finish all tasks, and it is commonly evaluated by calculating the time difference between the start and end times of the schedule. The tabular column displays the makespan values of various scheduling algorithms for a specific number of tasks. The different algorithms compared in the table are proposed: HESGA, G_SOS, ANN-BPSO, and MALO.

From Figure 2, we can see that as the number of tasks increases, the Makespan also increases for all the algorithms. However, certain algorithms perform better than others in terms of Makespan. MSA-CSA has the lowest Makespan for 1000, 2000, 3000, and 5000 tasks, with Makespans of 253, 652, 1056, and 1750 s, respectively. HESGA, G_SOS, and MALO have higher Makespans in all cases. For 4000 tasks, MSA-CSA has a Makespan of 1205 s, which is higher than ANN—BPSO's Makespan of 1500 s, making ANN—BPSO the better algorithm for this task size. Overall, the proposed MSA-CSA algorithm performs best among the five algorithms for task transfer and scheduling in CC, as it has the lowest Makespan for most task sizes. Figure 2 serves as evidence of the effectiveness of the MSA-CSA algorithm in minimizing Makespan and its superiority over HESGA, G_SOS, and MALO. ANN—BPSO performs better than MSA-CSA only for the 4000 tasks scenario. Hence, the proposed model is considered the best algorithm among the five for task transfer and task scheduling in CC.

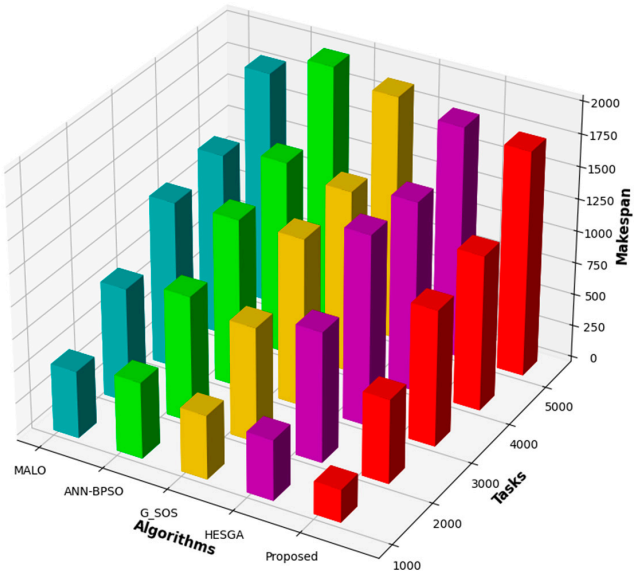

Figure 2. Comparative results based on Makespan.

5.4.2. Degree of Imbalance Result

The degree of imbalance refers to the difference in workload assigned to each computing resource (such as virtual machines) in a cloud system. An algorithm with a lower degree of imbalance distributes the workload more evenly, ensuring that no resource is overloaded while others remain underutilized. This is an important consideration in CC as it directly affects the performance and efficiency of the model.

The MSA-CSA algorithm has the lowest degree of imbalance compared to four other algorithms (HESGA, G_SOS, ANN-BPSO, and MALO) for different numbers of tasks (1000–5000) in cloud computing as can be seen in Figure 3. The DoI values are represented by a numerical value, where lower values indicate a more even workload distribution. The MSA-CSA algorithm has the lowest DoI for all task numbers, with 1.2 for 1000 tasks and 2.1 for 5000 tasks. HESGA has the second-lowest degree of imbalance, followed by G_SOS, ANN-BPSO, and MALO. This suggests that MSA-CSA may be a suitable algorithm for task transfer and scheduling in CC systems that require a more even workload distribution among computing resources.

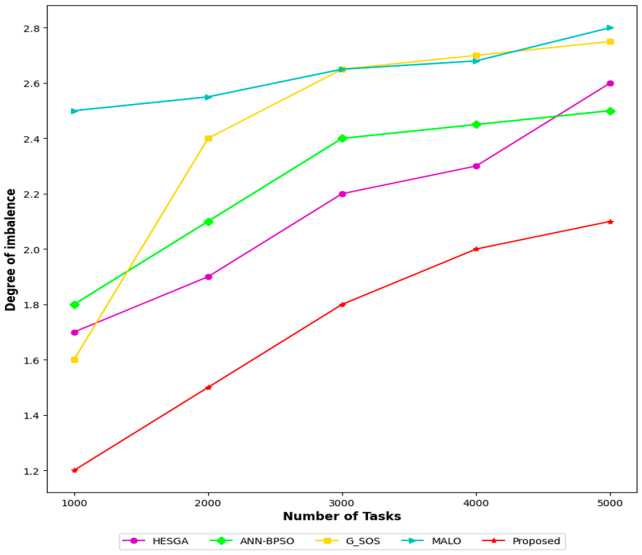

Figure 3. Comparative results based on DOI.

5.4.3. Resource Utilization Result

Resource utilization refers to the efficiency with which resources (such as computing power and memory) are utilized by an algorithm to complete a given task. A higher resource utilization indicates that the algorithm is using the available resources more effectively, leading to faster and more accurate task completion. According to Figure 4, the proposed MSA-CSA algorithm has the highest resource utilization for all task sizes, ranging from 96.20% for 5000 tasks to 98.70% for 1000 tasks. HESGA has the second-highest resource utilization, ranging from 95.20% for 5000 tasks to 97.20% for 1000 tasks. G_SOS has the third-highest resource utilization, ranging from 94.50% for 5000 tasks to 96.90% for 1000 tasks. ANN-BPSO has a lower resource utilization than the top three algorithms, ranging from 91.30% for 5000 tasks to 93.30% for 1000 tasks. MALO has the lowest resource utilization among the five algorithms, ranging from 94% for 5000 tasks to 95.45% for 1000 tasks. Overall, the MSA-CSA algorithm appears to be the most efficient in terms of resource utilization, followed by HESGA and G_SOS. However, it's worth noting that resource utilization is just one of many factors to consider when evaluating the performance of these algorithms, and other factors like accuracy and scalability should also be taken into account.

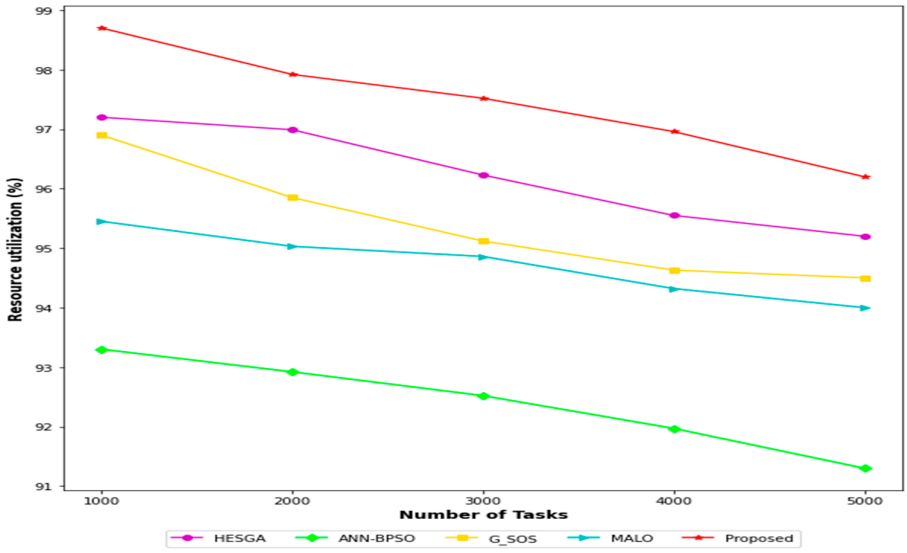

Figure 4. Comparative results based on resource utilization.

5.4.4. Average Waiting Time

The average waiting time represents the time a task spends in a queue waiting for its turn to be processed by the cloud system. It is an essential performance metric for evaluating task scheduling algorithms as it directly affects the system's overall throughput and user satisfaction. Looking at Figure 5, we can observe that the developed (MSA-CSA) algorithm has the lowest average waiting time for all task sizes, ranging from 1600 s for 1000 tasks to 6500 s for 5000 tasks. HESGA and G_SOS algorithms have comparable waiting times, with HESGA being slightly better than G_SOS for all task sizes. The ANN-BPSO and MALO algorithms have higher average waiting times than the other three algorithms, with MALO having the highest waiting time for all task sizes. In terms of average waiting time, the Proposed (MSA-CSA) algorithm outperforms the other algorithms evaluated in the study.

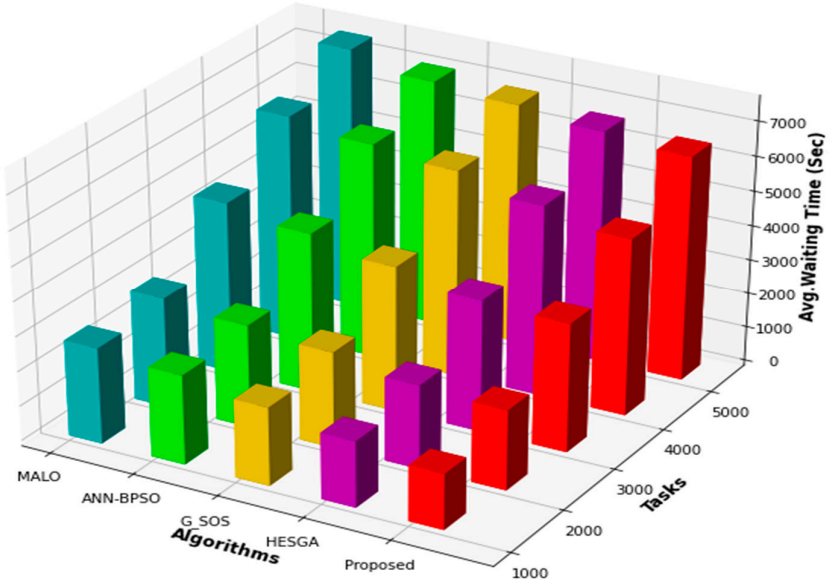

Figure 5. Comparative results based on AWT.

5.4.5. Cost Result

Figure 6 presents the number of tasks as the input for the algorithms and the corresponding cost output. Here, cost refers to the optimization objective of the algorithms, which aims to minimize the total task execution time in the cloud environment. Looking at the figure, we can observe that as the number of tasks increases, the cost for each algorithm also increases. However, some algorithms perform better than others in terms of cost. Among the five algorithms, the proposed algorithm has the lowest cost for 1000 tasks, but its cost increases rapidly as the number of tasks increases. For 1000 tasks, the HESGA algorithm has a marginally higher cost, but its performance remains consistent as the tasks rise. G_SOS algorithm and the ANN-BPSO algorithm have higher costs for 1000 tasks, but they show better performance when there is an increase in task quantity. Finally, the MALO has the highest cost among all the algorithms for all numbers of tasks.

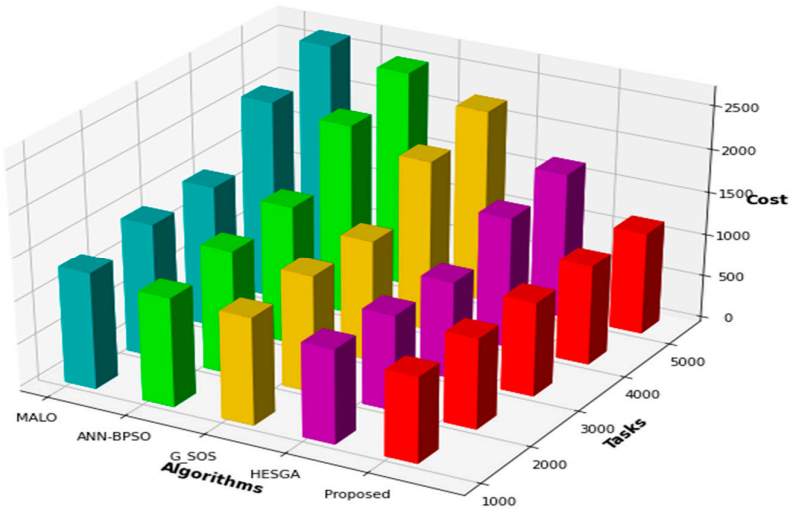

Figure 6. Comparative results based on cost.

5.4.6. Latency Result

Figure 7 displays the latency (in secs) for each algorithm for different numbers of tasks. For 1000 tasks, the MSA-CSA algorithm has the lowest latency at 75 s, followed by the HESGA algorithm at 81 s. For 5000 tasks, the MSA-CSA algorithm has the lowest

latency at 120 s, followed by the HESGA algorithm at 125 s. In general, the MSA-CSA algorithm performs the best in terms of minimizing latency, with the lowest latency for all task sizes. The HESGA algorithm is the second best, followed by the ANN-BPSO and MALO algorithms, which have similar performances. The G_SOS algorithm consistently has the highest latency.

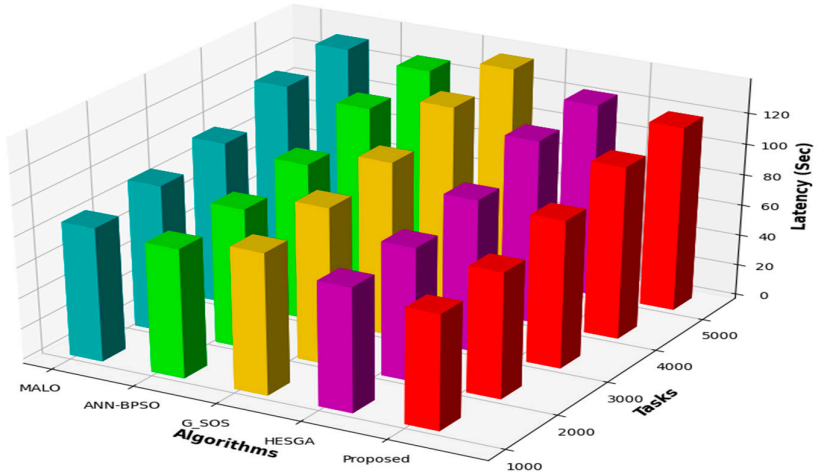

Figure 7. Comparative results based on Latency.

5.4.7. Execution Time

Figure 8 displays the execution time results for varying task sizes, indicating that execution time increases as the number of tasks increases for all algorithms. The MSA-CSA algorithm demonstrated the highest efficiency in terms of execution time for this particular problem, with the lowest execution time for all task sizes ranging from 2600 s for 1000 tasks to 9800 s for 5000 tasks. The other algorithms, HESGA, G_SOS, ANN-BPSO, and MALO, had higher execution times than MSA-CSA, with G_SOS and ANN-BPSO having the second and third-lowest execution times, respectively. In summary, the results of this study demonstrate that MSA-CSA is the most efficient algorithm in terms of execution time for task transfer and scheduling in the cloud environment. However, G_SOS and ANN-BPSO are also viable options for this problem, as they had lower execution times than HESGA and MALO. These findings may have important implications for the development of more efficient algorithms for task transfer and scheduling in the cloud.

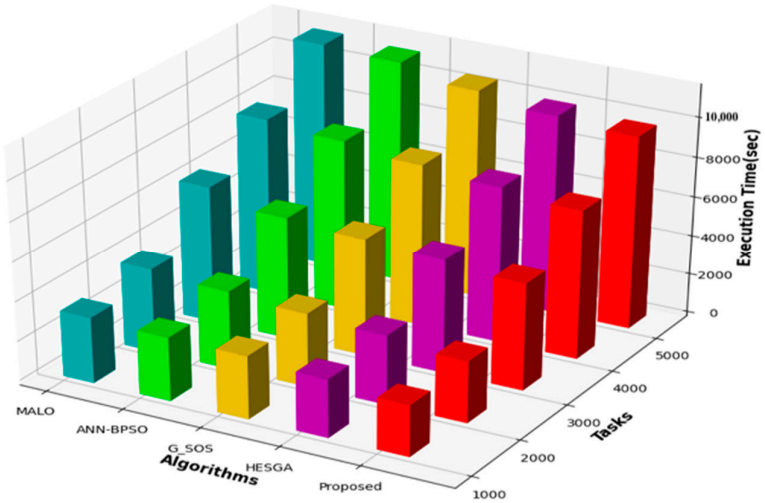

Figure 8. Comparative results based on execution time.

5.4.8. Bandwidth Utilization

Bandwidth utilization refers to the efficiency of utilizing the available bandwidth for transferring data between the different nodes in a cloud environment. It is a crucial metric in CC as it affects the overall performance and cost-effectiveness of cloud-based applications. Figure 9 shows the bandwidth utilization (%) achieved by each algorithm for different numbers of tasks, ranging from 1000 to 5000. The higher the percentage, the more efficiently the algorithm utilizes the available bandwidth for task transfer and scheduling. According to the figure, the proposed MSA-CSA algorithm achieves the highest bandwidth utilization for all numbers of tasks, ranging from 95.20% for 1000 tasks to 98.45% for 5000 tasks. The HESGA algorithm also achieves high bandwidth utilization, ranging from 94.50% for 1000 tasks to 97.81% for 5000 tasks. The G_SOS, ANN-BPSO, and MALO algorithms achieve lower bandwidth utilization than MSA-CSA and HESGA for all numbers of tasks. However, they still achieve reasonable bandwidth utilization ranging from 93.20% to 95.94%. In summary, the figure suggests that MSA-CSA and HESGA are the most effective algorithms for task transfer and task scheduling in the cloud in terms of bandwidth utilization, while G_SOS, ANN-BPSO, and MALO can also achieve reasonable bandwidth utilization but are less efficient than the former two algorithms.

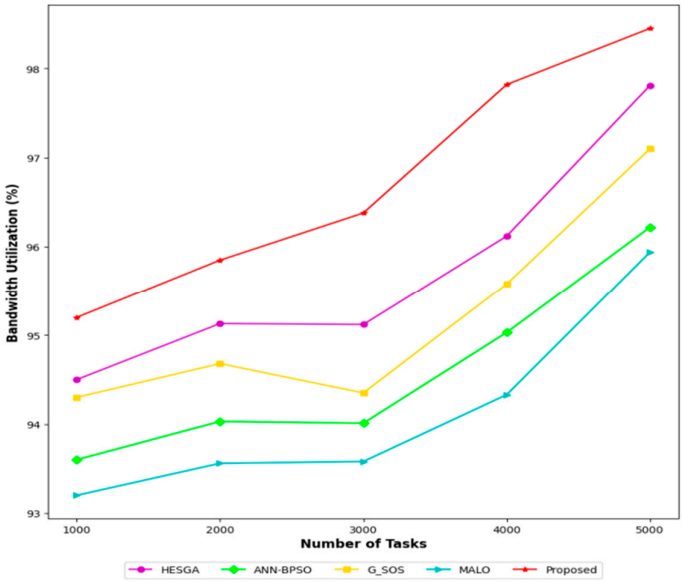

Figure 9. Comparative results based on Bandwidth utilization.

5.4.9. Response Time Result

Figure 10 compares the response time (in seconds) of different algorithms for task transfer and scheduling in CC across different numbers of tasks. For 1000 tasks, the MSA-CSA algorithm had the lowest response time of 1.3 s, followed by HESGA, G_SOS, ANN-BPSO, and MALO. As the number of tasks increased, the response time increased for all algorithms. For 2000 tasks, MSA-CSA still had the lowest response time of 1.9 s, followed by HESGA, G_SOS, ANN-BPSO, and MALO. For 3000 tasks, MSA-CSA still had the lowest response time of 2.3 s, followed by HESGA, G_SOS, ANN-BPSO, and MALO. For 4000 tasks, MSA-CSA had the lowest response time of 2.65 s, followed by HESGA, G_SOS, ANN-BPSO, and MALO. For 5000 tasks, MSA-CSA had the lowest response time of 3.5 s, followed by HESGA, G_SOS, ANN-BPSO, and MALO. Overall, MSA-CSA was found to be the most efficient algorithm for task transfer and scheduling in CC, with the Hybrid MSA and CSA having the lowest response time across all numbers of tasks. This indicates that MSA-CSA is the most efficient algorithm for task transfer and task scheduling in the cloud.

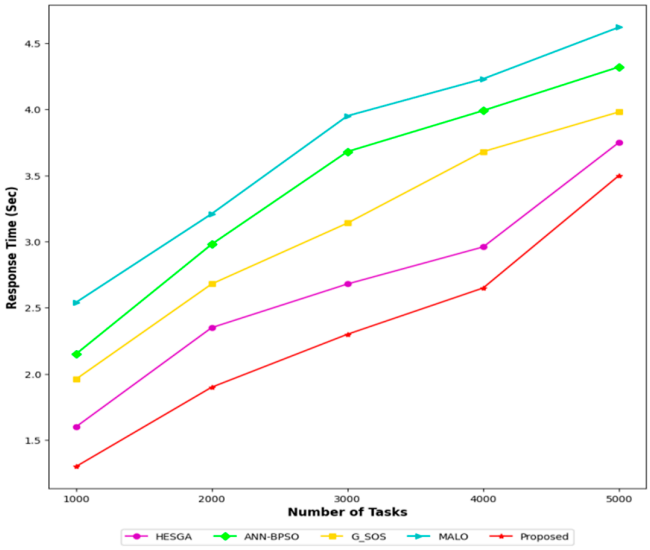

Figure 10. Comparative results based on response time.

5.4.10. Throughput Result

Throughput refers to the rate of successful task completions per unit of time. Figure 11 shows the throughput achieved by each algorithm for different numbers of tasks (ranging from 1000 to 5000). The results indicate that all algorithms achieve relatively high throughput, with all values above 90%. The MSA-CSA algorithm shows the highest throughput for all task numbers, ranging from 96% for 1000 tasks to 99.40% for 5000 tasks. The HESGA algorithm also performs well, achieving throughput values ranging from 93.36% to 96.80%. The G_SOS, ANN-BPSO, and MALO algorithms show slightly lower throughput values compared to MSA-CSA and HESGA but still achieve values above 90% for all task numbers. Overall, the results suggest that the proposed MSA-CSA algorithm and the HESGA algorithm are the most effective for task transfer and task scheduling in a cloud environment, as they achieve the highest throughput values across all task numbers.

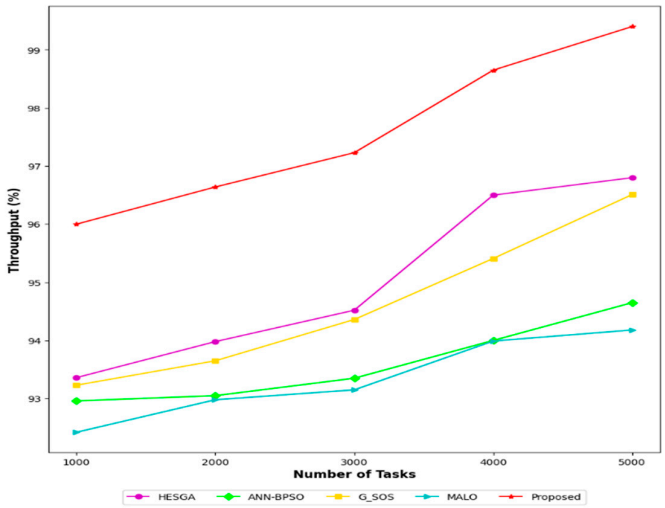

Figure 11. Comparative results based on throughput.

5.4.11. Speed

Table 6 shows the transfer speed performance for five different proposed task scheduling algorithms (HESGA, G_SOS, ANN-BPSO, and MALO) as compared to the proposed algorithm for varying numbers of tasks (1000, 2000, 3000, 4000, and 5000). The values in the table represent the percentage of the maximum transfer speed achieved by each algorithm.

For example, for 1000 tasks, the proposed algorithm achieved 55% of the maximum transfer speed, while HESGA achieved 42%, G_SOS achieved 38%, ANN-BPSO achieved 36%, and MALO achieved 31%. As the number of tasks increases, we can observe that the proposed algorithm achieves higher transfer speeds compared to the other algorithms. For example, when handling 5000 tasks, the proposed algorithm achieved 92% of the maximum transfer speed, while the other algorithms achieved between 66% and 82% of the maximum transfer speed. Overall, the table suggests that the proposed algorithm exhibits superior performance in transfer speed for task scheduling when dealing with a large number of tasks.

Table 6. Comparative analysis of transfer speed.

Tasks	Proposed	HESGA	G_SOS	ANN—BPSO	MALO
1000	55%	42%	38%	36%	31%
2000	65%	59%	53%	49%	42%
3000	78%	68%	64%	59%	51%
4000	85%	79%	71%	67%	64%
5000	92%	82%	74%	69%	66%

5.4.12. Security

Figure 12 displays the security percentage achieved by both algorithms when using different encryption techniques, such as DES, 3DES, and AES. The security percentage indicates the level of security achieved by the algorithm when encrypting and decrypting data. The higher the security percentage, the better the algorithm's ability to protect data from unauthorized access or theft. According to the figure, the proposed algorithm achieves a security percentage of 99.50%, indicating that it is highly secure and able to protect data from unauthorized access. The reason behind this significant percentage is that the proposed algorithm employs an integrated approach that combines the strengths of both MSA and CSA, allowing it to efficiently allocate and transfer tasks while maintaining a high level of security. The figure also shows that when using DES, the proposed algorithm achieves a security percentage of 98%, which is still considered highly secure but slightly lower than the overall percentage achieved by the algorithm. Similarly, the algorithm achieves a security percentage of 97.60% and 98.40% when using 3DES and AES, respectively. In conclusion, the figure suggests that the proposed hybrid moth swarm and Chameleon swarm algorithm is highly secure and effective in task transfer and task scheduling in the CC environment. Additionally, the algorithm's ability to achieve a high level of security when using different encryption techniques makes it a versatile solution for securing data in various scenarios.

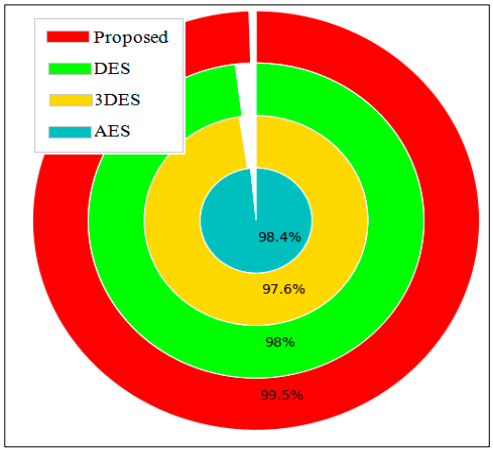

Figure 12. Comparative results based on security.

5.4.13. Result Based on the Fitness Function

1. Convergence trends for Makespan

The fitness function value-based comparison outcomes of the proposed and existing algorithms are presented in Figure 13. The figures indicate that the newly developed algorithm achieved superior performance compared to the existing one, as measured by the Fitness function. As illustrated in Figure 13a–e, the proposed algorithm achieved the lowest average values for the fitness function across all task cases.

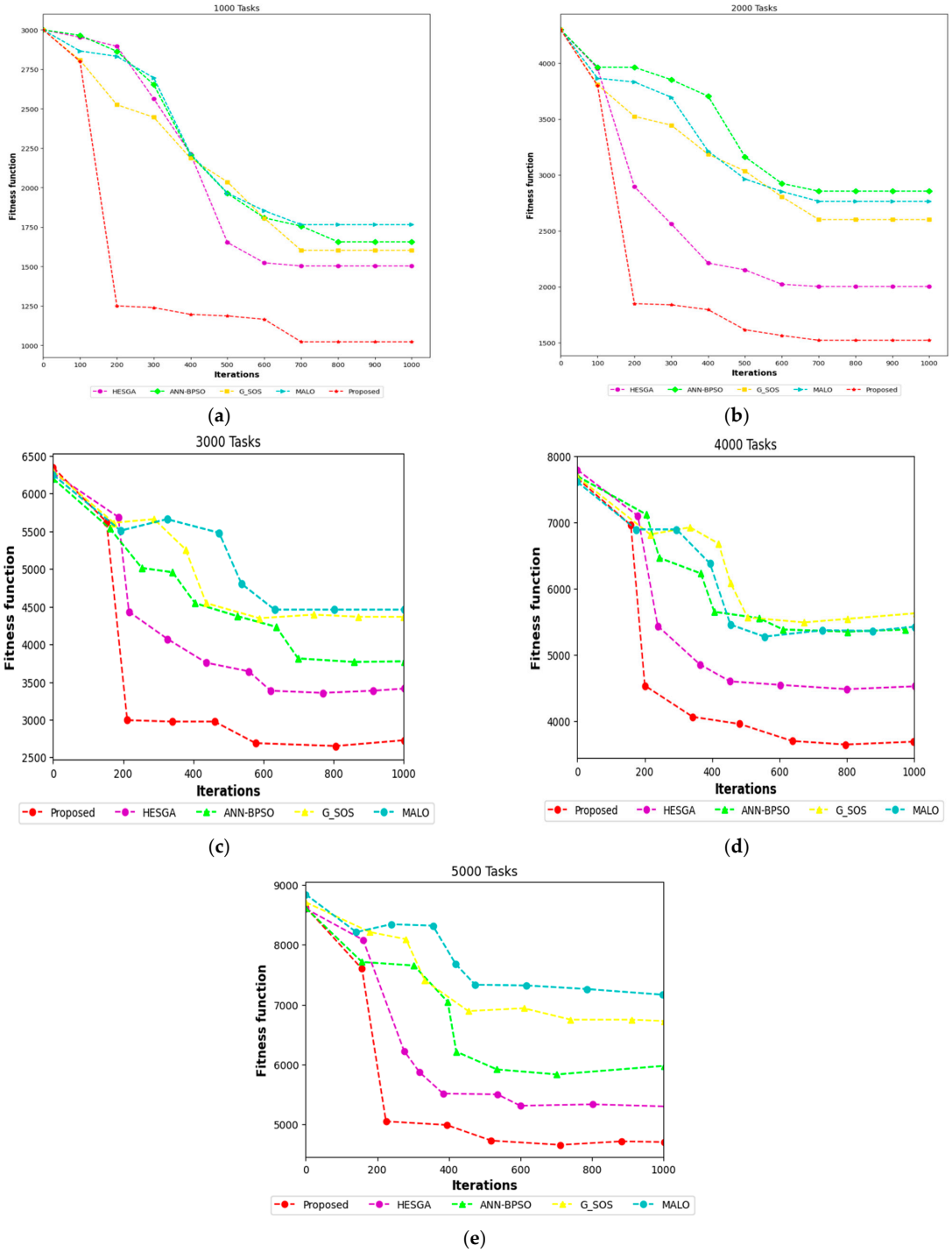

Figure 13. Comparative results according to fitness function for (**a**) 1000 tasks, (**b**) 2000 tasks, (**c**) 3000 tasks, (**d**) 4000 tasks, (**e**) 5000 tasks.

Additionally, the proposed algorithm's stability is evident as it effectively solves task scheduling problems of varying sizes. This validates the efficacy of using the proposed multi-objective function and hybrid version as a method to address task scheduling problems.

2. Convergence trends for Throughput

In Figure 14, the x-axis represents the number of iterations or generations, while the y-axis represents the throughput, indicating the rate of task processing per millisecond. As the algorithm progresses through the iterations, the line graph will illustrate how the throughput changes over time. Initially, the throughput values may start at relatively lower levels, but with each iteration, the algorithm improves task processing efficiency, leading to higher throughput values. The graph's trend line will indicate the convergence of throughput towards higher values, signifying the algorithm's ability to handle tasks more efficiently over iterations.

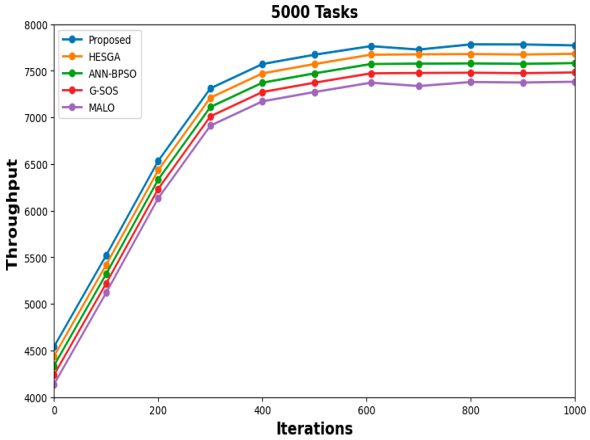

Figure 14. Convergence of throughput with 5000 tasks.

3. Convergence trends for Latency

In Figure 15, the x-axis represents the number of iterations or generations, while the y-axis represents the latency in milliseconds. As the algorithm progresses through iterations or generations, the line graph will show how the latency decreases over time. Initially, the latency may be relatively high, but as the algorithm iteratively optimizes the task-scheduling process, the latency gradually decreases. The graph's trendline will exhibit a downward trajectory, indicating that the algorithm is converging towards lower latency values. as the algorithm reaches closer to convergence, the line may flatten, indicating that further iterations have less impact on reducing latency.

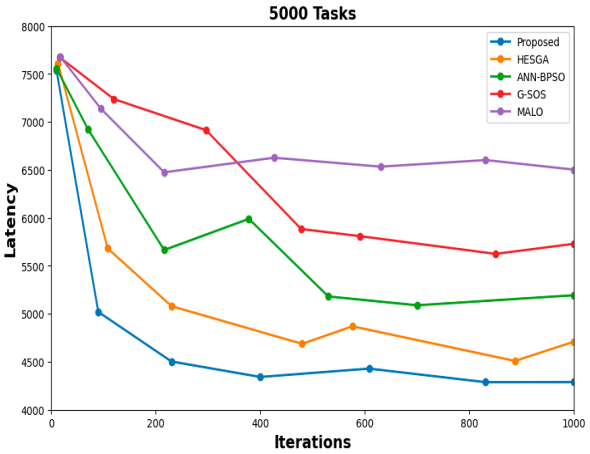

Figure 15. Convergence of latency with 5000 tasks.

From this experimental, additional configurations for the cloud environment are tested under different simulation parameters. From the above figure, we can effectively demonstrate the evolving performance of the proposed algorithm in terms of makespan, latency, and throughput with each iteration, offering a clear visualization of its iterative convergence trend in optimizing task scheduling in cloud computing.

4. Result analysis based on different types of tasks

Table 7 provides a tabular representation of the experimental results for the different task characteristics. It includes Task Type, Number of Tasks, Memory Requirement, CPU Requirement, Makespan, Throughput, and Latency.

Table 7. Performance Evaluation of Hybrid MSA-CSA for Task Scheduling in with Varying Task Characteristics and Workloads.

Task Type	No. of Tasks	Memory Requirement	CPU Requirement	Makespan (ms)	Throughput (tasks/ms)	Latency (ms)
Memory-Intensive	1000	High	Low	500	0.002	110
Memory-Intensive	2000	High	Low	480	0.003	120
Memory-Intensive	3000	High	Low	490	0.0025	125
Memory-Intensive	4000	High	Low	470	0.003	145
Memory-Intensive	5000	High	Low	480	0.002	155
CPU-Intensive	1000	Low	High	550	0.001	120
CPU-Intensive	2000	Low	High	530	0.002	125
CPU-Intensive	3000	Low	High	540	0.0018	135
CPU-Intensive	4000	Low	High	520	0.0015	140
CPU-Intensive	5000	Low	High	510	0.0012	150

For the Memory-Intensive tasks, characterized by high memory demands and low CPU requirements, the algorithm demonstrated efficient task scheduling. The makespan, representing the total time taken to complete all tasks, showed a decreasing trend as the number of tasks increased. For instance, with 1000 Memory-Intensive tasks, the makespan was 500 ms, while it decreased to 470 ms with 4000 tasks. The throughput, indicating the rate of task processing per millisecond, remained relatively stable at around 0.002 to 0.003 tasks/ms, reflecting consistent processing efficiency. The latency, representing the time delay between task submission and execution, showed a moderate increase in the number of tasks.

Similarly, for the CPU-Intensive tasks, characterized by low memory demands and high CPU requirements, the algorithm maintained efficient task scheduling. The makespan for CPU-intensive tasks exhibited slight fluctuations with varying workloads but generally remained low. For example, with 1000 CPU-intensive tasks, the makespan was 550 ms, while it decreased to 510 ms with 5000 tasks. The throughput gradually decreased as the number of tasks increased, indicating a subtle decline in processing efficiency. The latency for CPU-intensive tasks experienced a moderate increase with the number of tasks.

Overall, the experimental results demonstrate that the proposed hybrid algorithm is capable of optimizing task scheduling for both Memory-Intensive and CPU-intensive tasks in cloud computing. It efficiently handles varying workloads, achieving low makespan and latency while maintaining reasonable throughput for different task characteristics.

Figure 16 presents the CPU time (in seconds) for each algorithm's execution. Our proposed algorithm, which leverages a hybrid approach combining the Moth Swarm Algorithm (MSA) and Chameleon Swarm Algorithm (CSA) for task scheduling optimization, demonstrates impressive efficiency, requiring only 12 s of CPU time. The HESGA algorithm, integrating Harmony Search with the Elitist Group Algorithm, follows closely with 14 s of CPU time. Next, the ANN-BPSO algorithm, which employs an Artificial Neural Network with Binary Particle Swarm Optimization, takes approximately 55 s. G_SOS, a Genetic Algorithm with SOS, exhibits a CPU time of 60 s, while the MALO algorithm, employing a meta-heuristic Algorithm with Local Search Optimization, records a CPU time of 77 s. These results indicate the superiority of our proposed hybrid algorithm in terms of

computational efficiency, making it a promising solution for optimizing task scheduling in cloud computing environments.

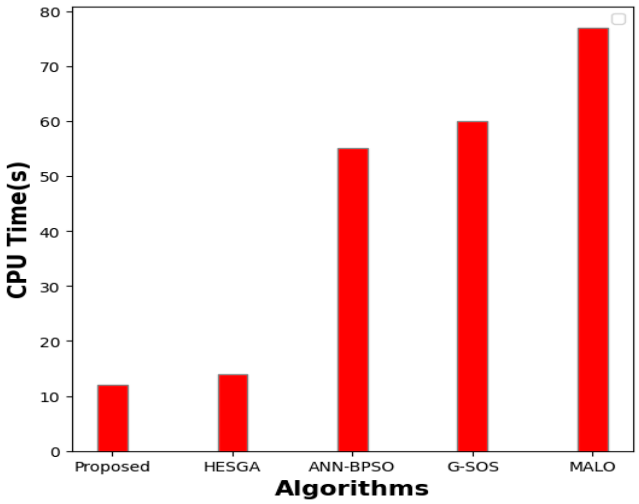

Figure 16. Comparison of CPU time for task scheduling algorithms.

Furthermore, to verify the superiority of the proposed hybrid algorithm, we compare it with existing models using the same parameter settings as described in the previous section. The experimental testing is performed in two different scenarios as follows:

Scenario 1

In this scenario, we have considered 50 virtual machines that are fixed in numbers, and each machine has the same processing capacity required to execute the tasks assigned to them. The set of randomly generated tasks whose number varied from 5000 to 10,000 in the interval are considered for the evaluation of the proposed technique. Each job size lies in the range of 500 million instructions to 50,000 million instructions. The results and analysis focus on three aspects: Execution time, response time, transfer speed, and fitness value.

Result analysis based on execution time in scenario 1

Table 8 summarizes execution times for different task quantities (ranging from 6000 to 10,000) using various task scheduling algorithms: Proposed, HESGA, G_SOS, ANN—BPSO, and MALO in scenario 1. Execution time is the duration (in units like seconds) taken to complete tasks. For instance, with 6000 tasks, the proposed algorithm took 2700 units, while HESGA, G_SOS, ANN—BPSO, and MALO took 3054, 3595, 3215, and 3725 secs, respectively. As task quantity increased to 10,000, execution times rose: Proposed took sec units, HESGA 10,275 units, G_SOS 10,562 units, ANN—BPSO 12,523 s, and MALO sec units. Lower execution times indicate higher efficiency and faster task completion, demonstrating algorithmic effectiveness in managing workload within cloud computing.

Table 8. Comparative results based on execution time.

Tasks	Proposed	HESGA	G_SOS	ANN—BPSO	MALO
6000	2700	3054	3595	3215	3725
7000	3426	3561	4025	3965	4553
8000	5623	5789	6264	6214	7254
9000	7964	8331	8236	9254	9362
10,000	9900	10,275	10,562	12,523	12,598

Result analysis based on response time in scenario 1

From Table 9, it is evident that the proposed algorithm consistently exhibits the lowest response times across all task quantities, ranging from 6000 to 10,000 tasks. For instance,

with 6000 tasks, the proposed algorithm demonstrates a response time of 1.5 s, compared to higher response times for other algorithms such as HESGA (2.45 s), G_SOS (2.87 s), ANN—BPSO (3.58 s), and MALO (5.75 s). Lower response times signify a more effective scheduling approach, enabling faster system responses to the assigned tasks. Therefore, based on this analysis, the proposed algorithm stands out for its effectiveness in minimizing response times and optimizing task scheduling within scenario 1.

Table 9. Comparative results based on response time.

Tasks	Proposed	HESGA	G_SOS	ANN—BPSO	MALO
6000	1.5	2.45	2.87	3.58	5.75
7000	2.5	3.78	3.58	4.75	6.85
8000	3.2	4.57	4.97	6.45	8.54
9000	4.2	6.8	8.7	8.65	10.22
10,000	5.1	8.9	10.7	11.54	13.54

Result analysis based on transfer speed in scenario 1

Table 10 clearly demonstrates the efficiency of the proposed algorithm in terms of transfer speed across varying task quantities in scenario 1. At each task quantity, the proposed algorithm consistently achieves the highest transfer speed compared to other algorithms. For instance, at 6000 tasks, the proposed algorithm achieves a transfer speed of 52%, surpassing HESGA (40%), G_SOS (37%), ANN—BPSO (34%), and MALO (29%). This pattern persists as the task quantity increases, reinforcing the efficiency of the Proposed algorithm in facilitating faster and more efficient data transfer. The higher transfer speeds attained by the Proposed algorithm signify its effectiveness in optimizing data transmission and, consequently, enhancing overall system efficiency. This observation underscores the viability and efficacy of the proposed model for efficient task scheduling and data transfer within the defined scenario.

Table 10. Comparative results based on transfer speed.

Tasks	Proposed	HESGA	G_SOS	ANN—BPSO	MALO
6000	52%	40%	37%	34%	29%
7000	62%	55%	52%	47%	39%
8000	75%	65%	60%	57%	49%
9000	84%	77%	69%	66%	62%
10,000	93%	79%	72%	62%	55%

Result analysis based on fitness value in scenario 1

Figure 17 encapsulates a comprehensive analysis based on fitness values within scenario 1, evaluating the efficiency and effectiveness of task scheduling algorithms. Fitness values serve as vital metrics representing how well each algorithm optimizes the given objective related to task scheduling. In the presented context, a higher fitness value signifies a more optimal solution. Notably, the proposed algorithm consistently demonstrates superior fitness values compared to HESGA, G_SOS, ANN—BPSO, and MALO for varying task quantities. This observation underscores the efficiency and effectiveness of the proposed algorithm in achieving optimal task scheduling outcomes. The higher fitness values achieved by the proposed algorithm emphasize its potential for enhancing task scheduling within the specified scenario, suggesting its viability for implementation and deployment.

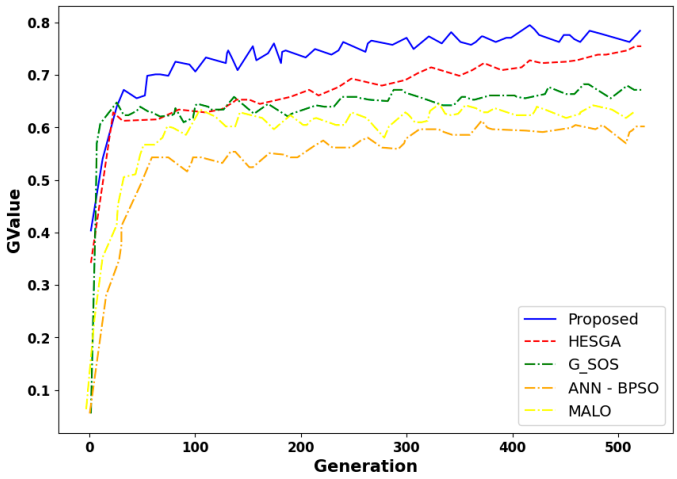

Figure 17. Comparative results based on fitness value in scenario 1.

Scenario 2

In this scenario, we choose N number of tasks and randomly select ten virtual machines (10, 35, 47, 60, 72, 85, 93, 100, 120 and 150).

Result analysis based on execution time in scenario 2

In Scenario 2, Table 11 presents an analysis of execution times for varying virtual machine serial numbers using different task scheduling algorithms: Proposed, HESGA, G_SOS, ANN—BPSO, and MALO. The focus is on elucidating the efficiency of the Proposed algorithm. Notably, for each virtual machine, the Proposed algorithm consistently yields the lowest execution times. This indicates the superior efficiency of the Proposed model in minimizing task execution durations across different virtual machine configurations. The trend persists consistently, showcasing the Proposed algorithm's effectiveness in optimizing task scheduling and execution, reinforcing its potential for enhancing overall system performance within Scenario 2.

Table 11. Comparison of execution time of VM with existing algorithms.

VM Serial Number	Proposed	HESGA	G_SOS	ANN—BPSO	MALO
10	2.5	3.7	4.5	4.2	3.4
35	2.7	4.5	4.8	4.6	4.5
47	3.2	4.9	5.78	5.4	5.34
60	3.75	5.2	6.43	5.78	6.72
72	4.23	5.9	6.75	6.95	7.23
85	4.75	6.3	7.25	7.8	7.89
93	4.9	6.9	7.5	8.45	8.45
100	5.24	7.4	8.54	9.25	9.43
120	5.62	7.8	9.12	10.15	10.76
150	5.8	8.4	10.2	12.4	13.54

Result analysis based on response time in scenario 2

In Scenario 2, the presented Table 12 provides a thorough analysis of response times concerning various virtual machine (VM) serial numbers. This analysis encompasses the utilization of distinct task scheduling algorithms, namely Proposed, HESGA, G_SOS, ANN—BPSO, and MALO. Of particular significance is the consistent observation that the Proposed algorithm consistently yields the lowest response times across the range of VM serial numbers considered. This consistent superiority underscores the efficiency and

effectiveness of the Proposed model in enabling rapid system responses to tasks distributed across different VM configurations.

Table 12. Comparison of response time of VM with existing algorithms.

VM Serial Number	Proposed	HESGA	G_SOS	ANN—BPSO	MALO
10	1.2	2.75	2.5	3.5	4.5
35	1.5	3.5	3.12	4.12	4.87
47	1.7	4.9	4.5	4.67	5.4
60	2.5	5.43	5.25	5.8	5.75
72	2.75	6.8	6.78	6.45	6.75
85	3.2	7.43	7.34	6.9	8.23
93	3.5	9.5	7.98	7.23	12.45
100	3.9	12.54	8.45	9.12	15.67
120	4.5	15.3	10.5	10.34	16.25
150	4.75	16.32	11.25	13.5	17

Result analysis based on transfer speed in scenario 2

In Scenario 2, Table 13 presents an analysis of transfer speeds for different virtual machine serial numbers using various task scheduling algorithms: Proposed, HESGA, G_SOS, ANN—BPSO, and MALO. Notably, the proposed algorithm consistently achieves the highest transfer speeds across all virtual machine configurations. This emphasizes the efficiency of the proposed model in facilitating faster data transfer within the system. The observed trend underscores the proposed algorithm's efficacy in optimizing data transmission, which is essential for enhancing overall system efficiency within Scenario 2.

Table 13. Comparison of transfer speed of VM with existing algorithms.

VM Serial Number	Proposed	HESGA	G_SOS	ANN—BPSO	MALO
10	55%	45%	34%	38%	31%
35	62%	54%	36%	42%	38%
47	68%	59%	39%	48%	49%
60	72%	62%	45%	52%	65%
72	76%	69%	49%	57%	69%
85	85%	72%	55%	63%	73%
93	89%	78%	59%	69%	78%
100	92%	81%	62%	72%	82%
120	94%	85%	68%	79%	85%
150	96%	88%	72%	82%	87%

Result analysis based on fitness value in scenario 2

Figure 18 illustrates that, for varying virtual machine serial numbers, the proposed algorithm consistently achieves superior fitness values compared to HESGA, G_SOS, ANN—BPSO, and MALO. This shows the efficiency and effectiveness of the proposed algorithm in optimizing task scheduling objectives, leading to a more optimal and efficient task distribution. The higher fitness values attributed to the proposed algorithm highlight its potential to enhance task scheduling efficiency within the defined scenario, emphasizing its viability and effectiveness for implementation.

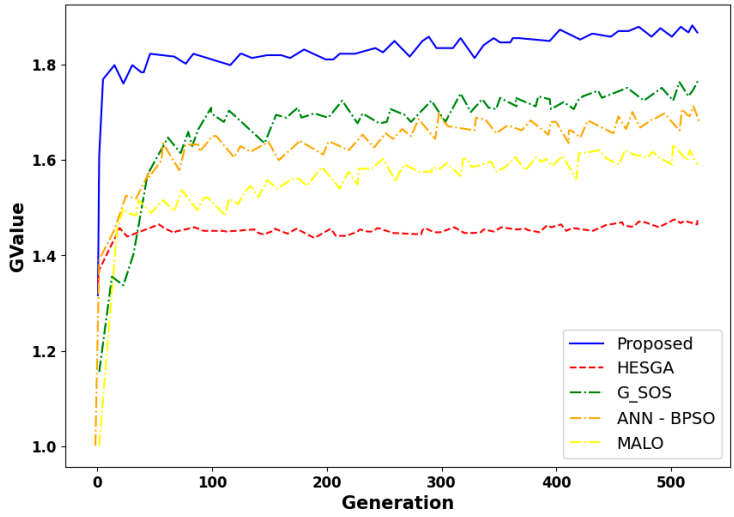

Figure 18. Comparative results based on fitness value in scenario 2.

In both Scenario 1 and Scenario 2, the proposed algorithm consistently demonstrated superior performance across various metrics. In Scenario 1, it showcased faster task execution (lower execution times), quicker system responsiveness (lower response times), and efficient data transfer (higher transfer speed). Moreover, it achieved higher fitness values, indicating optimal task scheduling. In Scenario 2, focusing on diverse virtual machine configurations, the proposed algorithm again outperformed others in execution time, response time, and transfer speed, affirming its efficiency and adaptability. These findings highlight the potential of the proposed algorithm to notably improve task scheduling efficiency and overall system performance, affirming its viability for practical integration within cloud computing environments.

6. Conclusions and Future Work

Cloud task scheduling is an essential aspect of cloud computing, and optimizing data transfer is crucial for delivering services at the right time. While single-objective cloud task scheduling has been extensively researched, multi-objective scheduling problems have recently gained attention. This paper proposes hybrid MSA-CSA algorithms that can effectively schedule cloud tasks and optimize bandwidth allocation, which helps reduce network congestion, prevent bottlenecks, and improve system performance. Optimizing makespan, throughput, execution time, cost, and latency in task scheduling is critical for efficient data transfer and scheduling in the cloud. MSA and CSA are combined due to their well-established effectiveness in optimization and task scheduling domains. The MSA is inspired by the natural behavior of moths and has shown promise in optimizing complex problems by simulating the moths' movement patterns and light attraction behaviour. On the other hand, the CSA is inspired by the adaptive nature of chameleons and incorporates adaptive strategies in the optimization process. Both algorithms offer unique characteristics such as exploration-exploitation balance, adaptability, and the ability to handle high-dimensional and non-linear optimization problems. By utilizing these algorithms in task scheduling, the authors aimed to harness their strengths in optimizing the allocation of computational tasks in cloud environments, ultimately enhancing task scheduling efficiency and overall system performance. The implemented polymorphic version of the Advanced Encryption Standard, P-AES, offers data security, improving the performance of the CC system. The experiment carried out using Python demonstrates that the newly developed algorithm ensures stability and efficiency in secure task scheduling. Therefore, the proposed approach can help cloud providers allocate resources effectively and prioritize data transfers based on their importance and urgency. Overall, this paper offers valuable insights into multi-objective cloud task scheduling and proposes effective algorithms for optimizing cloud task scheduling, data transfer, and data security. In the future, the

energy consumption optimization of CC data centers will be further analyzed, and effective combinations of AI technology and task scheduling algorithms will be comprehensively studied. Furthermore, the space complexity and time complexity of the proposed model will be elaborated. Apart from that, the proposed algorithm can be utilized to solve some other problems such as prediction model [42–44], reliability constraints problem [45], resource allocation [46], networks system [47,48], and consensus model [49] under the diverse environment.

Author Contributions: Conceptualization, S.A., H.G. and A.A.; methodology, S.A., H.G. and A.A.; software, S.A., H.G. and A.A.; validation, S.A., H.G. and A.A.; formal analysis, S.A., H.G. and A.A.; investigation, S.A., H.G. and A.A.; resources, S.A., H.G. and A.A.; data curation, S.A., H.G. and A.A.; writing—original draft preparation, S.A., H.G. and A.A.; writing—review and editing, S.A., H.G. and A.A.; visualization, S.A., H.G. and A.A.; supervision, S.A., H.G. and A.A.; project administration, S.A., H.G. and A.A.; funding acquisition, S.A., H.G. and A.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Deputyship for Research and Innovation, Ministry of Education in Saudi Arabia, grant number (IF2/PSAU/2022/01/22904).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors extend their appreciation to the Deputyship for Research and Innovation, Ministry of Education in Saudi Arabia, for funding this research work through the project number (IF2/PSAU/2022/01/22904).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Badri, S.; Alghazzawi, D.M.; Hasan, S.H.; Alfayez, F.; Hasan, S.H.; Rahman, M.; Bhatia, S. An Efficient and Secure Model Using Adaptive Optimal Deep Learning for Task Scheduling in Cloud Computing. *Electronics* **2023**, *12*, 1441. [CrossRef]
2. Khan, M.S.A.; Santhosh, R. Task scheduling in cloud computing using hybrid optimization algorithm. *Soft Comput.* **2022**, *26*, 13069–13079. [CrossRef]
3. Najafizadeh, A.; Salajegheh, A.; Rahmani, A.M.; Sahafi, A. Multi-objective Task Scheduling in cloud-fog computing using goal programming approach. *Clust. Comput.* **2022**, *25*, 141–165. [CrossRef]
4. Mangalampalli, S.; Karri, G.R.; Elngar, A.A. An Efficient Trust-Aware Task Scheduling Algorithm in Cloud Computing Using Firefly Optimization. *Sensors* **2023**, *23*, 1384. [CrossRef] [PubMed]
5. Xia, X.; Qiu, H.; Xu, X.; Zhang, Y. Multi-objective workflow scheduling based on genetic algorithm in cloud environment. *Inf. Sci.* **2022**, *606*, 38–59. [CrossRef]
6. Wang, X.; Yao, W. A Discrete Particle Swarm Optimization Algorithm for Dynamic Scheduling of Transmission Tasks. *Appl. Sci.* **2023**, *13*, 4353. [CrossRef]
7. Shao, K.; Song, Y.; Wang, B. PGA: A New Hybrid PSO and GA Method for Task Scheduling with Deadline Constraints in Distributed Computing. *Mathematics* **2023**, *11*, 1548. [CrossRef]
8. Bal, P.K.; Mohapatra, S.K.; Das, T.K.; Srinivasan, K.; Hu, Y.C. A joint resource allocation, security with efficient task scheduling in cloud computing using hybrid machine learning techniques. *Sensors* **2022**, *22*, 1242. [CrossRef]
9. Fu, X.; Sun, Y.; Wang, H.; Li, H. Task scheduling of cloud computing based on hybrid particle swarm algorithm and genetic algorithm. *Clust. Comput.* **2021**, *26*, 2479–2488. [CrossRef]
10. Rana, N.; Abd Latiff, M.S.; Abdulhamid, S.I.M.; Misra, S. A hybrid whale optimization algorithm with differential evolution optimization for multi-objective virtual machine scheduling in cloud computing. *Eng. Optim.* **2022**, *54*, 1999–2016. [CrossRef]
11. Pirozmand, P.; Jalalinejad, H.; Hosseinabadi, A.A.R.; Mirkamali, S.; Li, Y. An improved particle swarm optimization algorithm for task scheduling in cloud computing. *J. Ambient. Intell. Humaniz. Comput.* **2023**, *14*, 4313–4327. [CrossRef]
12. Naik, B.B.; Singh, D.; Samaddar, A.B. FHCS: Hybridised optimisation for virtual machine migration and task scheduling in cloud data center. *IET Commun.* **2020**, *14*, 1942–1948. [CrossRef]
13. Kakkottakath Valappil Thekkepuryil, J.; Suseelan, D.P.; Keerikkattil, P.M. An effective meta-heuristic based multi-objective hybrid optimization method for workflow scheduling in cloud computing environment. *Clust. Comput.* **2021**, *24*, 2367–2384. [CrossRef]
14. Huang, Y.; Zhang, S.; Wang, B. An Improved Genetic Algorithm with Swarm Intelligence for Security-Aware Task Scheduling in Hybrid Clouds. *Electronics* **2023**, *12*, 2064. [CrossRef]

15. Iranmanesh, A.; Naji, H.R. DCHG-TS: A deadline-constrained and cost-effective hybrid genetic algorithm for scientific workflow scheduling in cloud computing. *Clust. Comput.* **2021**, *24*, 667–681. [CrossRef]
16. Masadeh, R.; Alsharman, N.; Sharieh, A.; Mahafzah, B.A.; Abdulrahman, A. Task scheduling on cloud computing based on sea lion optimization algorithm. *Int. J. Web Inf. Syst.* **2021**, *17*, 99–116. [CrossRef]
17. Natesan, G.; Chokkalingam, A. An improved grey wolf optimization algorithm based task scheduling in cloud computing environment. *Int. Arab J. Inf. Technol.* **2020**, *17*, 73–81. [CrossRef] [PubMed]
18. Huang, H.; Cuan, X.; Chen, Z.; Zhang, L.; Chen, H. A Multiregional Agricultural Machinery Scheduling Method Based on Hybrid Particle Swarm Optimization Algorithm. *Agriculture* **2023**, *13*, 1042. [CrossRef]
19. Mohammadzadeh, A.; Masdari, M.; Gharehchopogh, F.S.; Jafarian, A. A hybrid multi-objective metaheuristic optimization algorithm for scientific workflow scheduling. *Clust. Comput.* **2021**, *24*, 1479–1503. [CrossRef]
20. Dubey, K.; Sharma, S.C. A novel multi-objective CR-PSO task scheduling algorithm with deadline constraint in cloud computing. *Sustain. Comput. Inform. Syst.* **2021**, *32*, 100605. [CrossRef]
21. Kaur, A.; Kumar, S.; Gupta, D.; Hamid, Y.; Hamdi, M.; Ksibi, A.; Elmannai, H.; Saini, S. Algorithmic Approach to Virtual Machine Migration in Cloud Computing with Updated SESA Algorithm. *Sensors* **2023**, *23*, 6117. [CrossRef] [PubMed]
22. Rajakumari, K.; Kumar, M.V.; Verma, G.; Balu, S.; Sharma, D.K.; Sengan, S. Fuzzy Based Ant Colony Optimization Scheduling in Cloud Computing. *Comput. Syst. Sci. Eng.* **2022**, *40*, 581–592. [CrossRef]
23. Attiya, I.; Abualigah, L.; Alshathri, S.; Elsadek, D.; Abd Elaziz, M. Dynamic jellyfish search algorithm based on simulated annealing and disruption operators for global optimization with applications to cloud task scheduling. *Mathematics* **2022**, *10*, 1894. [CrossRef]
24. Sharma, M.; Garg, R. HIGA: Harmony-inspired genetic algorithm for rack-aware energy-efficient task scheduling in cloud data centers. *Eng. Sci. Technol. Int. J.* **2020**, *23*, 211–224. [CrossRef]
25. Noorian Talouki, R.; Hosseini Shirvani, M.; Motameni, H. A hybrid meta-heuristic scheduler algorithm for optimization of workflow scheduling in cloud heterogeneous computing environment. *J. Eng. Des. Technol.* **2022**, *20*, 1581–1605. [CrossRef]
26. Nabi, S.; Ahmad, M.; Ibrahim, M.; Hamam, H. AdPSO: Adaptive PSO-based task scheduling approach for cloud computing. *Sensors* **2022**, *22*, 920. [CrossRef] [PubMed]
27. Zubair, A.A.; Razak, S.A.; Ngadi, M.A.; Al-Dhaqm, A.; Yafooz, W.M.; Emara, A.H.M.; Saad, A.; Al-Aqrabi, H. A Cloud Computing-Based Modified Symbiotic Organisms Search Algorithm (AI) for Optimal Task Scheduling. *Sensors* **2022**, *22*, 1674. [CrossRef]
28. Gupta, S.; Iyer, S.; Agarwal, G.; Manoharan, P.; Algarni, A.D.; Aldehim, G.; Raahemifar, K. Efficient prioritization and processor selection schemes for heft algorithm: A makespan optimizer for task scheduling in cloud environment. *Electronics* **2022**, *11*, 2557. [CrossRef]
29. Amer, D.A.; Attiya, G.; Zeidan, I.; Nasr, A.A. Elite learning Harris hawks optimizer for multi-objective task scheduling in cloud computing. *J. Supercomput.* **2022**, *78*, 2793–2818. [CrossRef]
30. Alboaneen, D.; Tianfield, H.; Zhang, Y.; Pranggono, B. A metaheuristic method for joint task scheduling and virtual machine placement in cloud data centers. *Future Gener. Comput. Syst.* **2021**, *115*, 201–212. [CrossRef]
31. Albert, P.; Nanjappan, M. WHOA: Hybrid based task scheduling in cloud computing environment. *Wirel. Pers. Commun.* **2021**, *121*, 2327–2345. [CrossRef]
32. Alsadie, D. A metaheuristic framework for dynamic virtual machine allocation with optimized task scheduling in cloud data centers. *IEEE Access* **2021**, *9*, 74218–74233. [CrossRef]
33. Agarwal, M.; Srivastava, G.M.S. Opposition-based learning inspired particle swarm optimization (OPSO) scheme for task scheduling problem in cloud computing. *J. Ambient. Intell. Humaniz. Comput.* **2021**, *12*, 9855–9875. [CrossRef]
34. Wei, X. Task scheduling optimization strategy using improved ant colony optimization algorithm in cloud computing. *J. Ambient. Intell. Humaniz. Comput.* **2020**, 1–12. [CrossRef]
35. Ramasamy, V.; Thalavai Pillai, S. An effective HPSO-MGA optimization algorithm for dynamic resource allocation in cloud environment. *Clust. Comput.* **2020**, *23*, 1711–1724. [CrossRef]
36. Altigani, A.; Hasan, S.; Barry, B.; Naserelden, S.; Elsadig, M.A.; Elshoush, H.T. A polymorphic advanced encryption standard–A novel approach. *IEEE Access* **2021**, *9*, 20191–20207. [CrossRef]
37. Luo, Q.; Yang, X.; Zhou, Y. Nature-inspired approach: An enhanced moth swarm algorithm for global optimization. *Math. Comput. Simul.* **2019**, *159*, 57–92. [CrossRef]
38. Braik, M.S. Chameleon Swarm Algorithm: A bio-inspired optimizer for solving engineering design problems. *Expert Syst. Appl.* **2021**, *174*, 114685. [CrossRef]
39. Velliangiri, S.; Karthikeyan, P.; Xavier, V.A.; Baswaraj, D. Hybrid electro search with genetic algorithm for task scheduling in cloud computing. *Ain Shams Eng. J.* **2021**, *12*, 631–639. [CrossRef]
40. Alghamdi, M.I. Optimization of Load Balancing and Task Scheduling in Cloud Computing Environments Using Artificial Neural Networks-Based Binary Particle Swarm Optimization (BPSO). *Sustainability* **2022**, *14*, 11982. [CrossRef]
41. Abualigah, L.; Diabat, A. A novel hybrid antlion optimization algorithm for multi-objective task scheduling problems in cloud computing environments. *Clust. Comput.* **2021**, *24*, 205–223. [CrossRef]
42. Li, B.; Tan, Y.; Wu, A.; Duan, G. A distributionally robust optimization based method for stochastic model predictive control. *IEEE Trans. Autom. Control* **2021**, *67*, 5762–5776. [CrossRef]

43. Zheng, Y.; Lv, X.; Qian, L.; Liu, X. An Optimal BP Neural Network Track Prediction Method Based on a GA and ACO Hybrid Algorithm. *J. Mar. Sci. Eng.* **2022**, *10*, 1399. [CrossRef]
44. Qian, L.; Zheng, Y.; Li, L.; Ma, Y.; Zhou, C.; Zhang, D. A New Method of Inland Water Ship Trajectory Prediction Based on Long Short-Term Memory Network Optimized by Genetic Algorithm. *Appl. Sci.* **2022**, *12*, 4073. [CrossRef]
45. Ma, K.; Li, Z.; Liu, P.; Yang, J.; Geng, Y.; Yang, B.; Guan, X. Reliability-Constrained Throughput Optimization of Industrial Wireless Sensor Networks With Energy Harvesting Relay. *IEEE Internet Things J.* **2021**, *8*, 13343–13354. [CrossRef]
46. Cao, B.; Sun, Z.; Zhang, J.; Gu, Y. Resource Allocation in 5G IoV Architecture Based on SDN and Fog-Cloud Computing. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 3832–3840. [CrossRef]
47. Xiao, Z.; Shu, J.; Jiang, H.; Lui, J.C.S.; Min, G.; Liu, J.; Dustdar, S. Multi-Objective Parallel Task Offloading and Content Caching in D2D-aided MEC Networks. *IEEE Trans. Mob. Comput.* **2022**, *22*, 6599–6615. [CrossRef]
48. Rashid, M.; Abed, W. IoT Sensor network data processing using the TWLGA scheduling algorithm and the Hadoop cloud platform. *Wasit J. Comp. Math.Sci.* **2023**, *2*, 135–145. [CrossRef]
49. Ma, J.; Hu, J. Safe consensus control of cooperative-competitive multi-agent systems via differential privacy. *Kybernetika* **2022**, *58*, 426–439. [CrossRef]