Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.058115

ARTICLE

A Multi-Objective Clustered Input Oriented Salp Swarm Algorithm in Cloud Computing

Juliet A. Murali[1,*] **and Brindha T.**[2]

[1]Department of Computer Science and Engineering, Noorul Islam Centre for Higher Education, Kumaracoil, 629180, India

[2]Department of Information Technology, Noorul Islam Centre for Higher Education, Kumaracoil, 629180, India

*Corresponding Author: Juliet A. Murali. Email: julietamurali@gmail.com

Received: 04 September 2024 Accepted: 01 November 2024 Published: 19 December 2024

ABSTRACT

Infrastructure as a Service (IaaS) in cloud computing enables flexible resource distribution over the Internet, but achieving optimal scheduling remains a challenge. Effective resource allocation in cloud-based environments, particularly within the IaaS model, poses persistent challenges. Existing methods often struggle with slow optimization, imbalanced workload distribution, and inefficient use of available assets. These limitations result in longer processing times, increased operational expenses, and inadequate resource deployment, particularly under fluctuating demands. To overcome these issues, a novel Clustered Input-Oriented Salp Swarm Algorithm (CIOSSA) is introduced. This approach combines two distinct strategies: Task Splitting Agglomerative Clustering (TSAC) with an Input Oriented Salp Swarm Algorithm (IOSSA), which prioritizes tasks based on urgency, and a refined multi-leader model that accelerates optimization processes, enhancing both speed and accuracy. By continuously assessing system capacity before task distribution, the model ensures that assets are deployed effectively and costs are controlled. The dual-leader technique expands the potential solution space, leading to substantial gains in processing speed, cost-effectiveness, asset efficiency, and system throughput, as demonstrated by comprehensive tests. As a result, the suggested model performs better than existing approaches in terms of makespan, resource utilisation, throughput, and convergence speed, demonstrating that CIOSSA is scalable, reliable, and appropriate for the dynamic settings found in cloud computing.

KEYWORDS

Cloud computing; clustering; resource allocation; scheduling; swam algorithms; optimization common within the subject discipline

1 Introduction

The advent of cloud computing has changed the ways and means through which firms deploy computational resources. Through its cloud computing services, it avails important services like memory, virtual processors, and storage based on a flexible on-demand basis. In general, virtual machines offer end users the opportunity to execute their applications and manage their workloads without the purchase of and maintenance of physical hardware. It is possible through the virtualization process to operate various virtual machines on one and only one physical server which helps improve

the use of resources [1]. The underlying physical infrastructures in the data centers are managed by the cloud service provider, allowing users to interact entirely with virtual resources through APIs or self-service dashboards.

Cloud computing is categorized into three major categories of services. These categories include IaaS, PaaS, and SaaS. The IaaS category forms the foundation, providing virtualized resources and storage solutions combined with networking capabilities. While PaaS and SaaS tend to abstract the complexities associated with managing infrastructure, IaaS allows direct access to underlying computing resources for even more flexibility and scalability. Here, based on changing requirements, assets are better provisioned and managed dynamically. Here, virtualization is quite easily achieved by allowing several VMs to run on the same physical host. This approach optimizes the usage of resources and allows for adjustments based on varying workloads without significant initial investments in hardware. Furthermore, IaaS provides users with greater control over their operating systems and applications while transferring the responsibility of managing the physical infrastructure to the provider [2,3]. PaaS, on the other hand, offers an environment that is already equipped with tools for creating and implementing applications, so removing the user from worries about the underlying infrastructure. SaaS provides software programs via the Internet, negating the need for the user to manage infrastructure in any way [4–6]. Without requiring software management or infrastructure supervision, the paradigm guarantees that applications are hosted and maintained by the service provider and are available to clients online.

Cloud providers integrate management models with clustering techniques that minimize resources wasted and enhance the service qualities to handle resource allocation optimally following the demand of the customer. One of the toughest challenges dealing with task scheduling in cloud environments falls directly on the efficiency of resource allocation among the clustered systems [7]. Effective scheduling is central to the maximization of available resources, especially in widespread architectures of the clouds where extensive tasks compete for limited amounts of processing capabilities. The primary concern when scheduling jobs is to minimize the makespan, which is the total time it takes to process all the scheduled jobs. It is also concerned with the minimization of costs due to resource usage. On the other hand, this problem is NP-hard and, therefore cannot find an exact solution for a very large problem in practical time [8–10]. Heuristics and metaheuristics are used to get as near to optimal solutions while dealing with such complexity. Among them, the applications of swarm intelligence algorithms have also gained considerable attention since they are capable of efficiently searching large solution spaces and producing high-quality scheduling solutions [11].

The three main algorithms in this area are MGWO, PSO, and SSO. The aforementioned techniques are based on cooperative search strategies that are modeled after collective natural phenomena, such as schools of fish or flocks of birds. PSO, for example, models particle behavior in a search space and modifies each particle's position according to its own experience as well as the experiences of the particles next to it. Similarly, MGWO uses a hierarchical structure to make judgments and draws inspiration from the hunting techniques adopted by grey wolves [12–14].

To optimize resource distribution, cloud computing systems employ a variety of management strategies, such as centralized as well as hierarchical types of management [15]. It allows easier control over the system since the resources are dealt with from a single point of control. However, it tends to cause bottlenecks in large systems. This type of model allows the increase in scalability and strength because, in this architecture, hierarchical types of management are distributed across many layers. Clustering Methods: This sub-categorized into partitional clustering example K-Means as well as hierarchical. While hierarchic classification has further been subdivided to include agglomerative

methods and divisive techniques. Hierarchical clustering divides the data points by two techniques: bottom-up, based on similarity to merge clusters and top-down, recurrent division of clusters. The other is partitional clustering: it divides the data into many predefined clusters by optimizing resource distribution in some real-time or huge scenarios such as resource location in a network environment or task allocation in a parallel environment [16,17]. Implementing these strategies would save the costs for cloud providers and minimize makespan, and therefore improve the overall efficiency of resource distribution, with dramatically improved performance in cloud environments. Major contributions are given below:

- To optimize resource allocation in cloud computing by prioritizing task distribution based on urgency and computational requirements, a new algorithm namely the Task Splitting Agglomerative Clustering (TSAC), is introduced, which enhances workload management through effective clustering, leading to improved processing efficiency and reduced makespan.
- To optimize resource allocation in cloud computing by employing a swarm intelligence-based approach, an algorithm named the Input Oriented Salp Swarm Algorithm (IOSSA) is integrated, which effectively manages resource deployment to meet varying workloads, thereby improving system throughput and reducing operational costs.

2 Literature Survey

Some existing methodologies proposed for resource allocation are reviewed as follows: A swarm-based meta-heuristic technique is the Generalized Ant Colony Optimizer developed by Kumar et al. [18]. GACO has emerged as a hybrid technique that combines the principles of Simple Ant Colony Optimization and Global Colony Optimization. GACO has its basis in the foraging behavior of ants. It was implemented to work in four different phases: establishment of a new colony, nearest food position search, solution balancer, and pheromone update. The performance of GACO is analyzed on seventeen well-reputed standard benchmark functions and compared with three different metaheuristic algorithms: The Genetic Algorithm, Particle Swarm Optimization, and Artificial Bee Colony. However, the limitation involves the fact that it may lead to degraded performance of GACO because of the complexity and dimensionality of the benchmark functions, hence defeating the purpose in some cases.

Singhal et al. [19], in turn, worked out the algorithm for the virtual machines' job scheduling problem based on the nature-inspired meta-heuristic approach. Thus, for problems faced by cloud service providers in cloud computing—energy consumptions and resulting costs-the Rock Hyrax Optimization (RHO) algorithm was developed. The idea of the algorithm was to minimize the overall makespan time and provide maximum energy efficiency by providing available resources to the jobs according to their loads. Two inferences were considered for the performance evaluation. First, the number of jobs was varied from 10 to 100 in steps of 10 for a given number of virtual machines. These scenarios would vary the input of jobs and resources dynamically and measure performance effectively under the algorithm. In such a way, the performance could deteriorate when considering larger and more complex cloud environments. It would, in turn, have adverse effects on scalability and efficiency within the algorithm.

Jena et al. [20] presented a unique approach to dynamically balance the load across virtual machines by hybridizing an enhanced Q-learning algorithm, called QMPSO, with a modified Particle Swarm Optimisation (MPSO) algorithm. Using the g_{best} and p_{best} values in accordance with the best action produced by the upgraded Q-learning algorithm, the hybridization procedure was carried out to modify the MPSO's velocity. The principal aim of this hybridization process was to improve virtual

machine performance by load balancing, throughput maximization, and task priority maintenance through task waiting time optimization. However, in extremely diverse cloud systems, where task kinds and resource capabilities vary, its efficacy may decline due to inadequate load balancing and decreased performance.

In order to solve the issue of work scheduling in a cloud computing environment, Hamed et al. [21] presented a unique solution called the Efficient Cooperation Search Algorithm (ECSA), which is based on the Cooperation Search Algorithm. There were just a few completely linked heterogeneous processors in the system. The makespan, speedup, efficiency, and throughput of the method were compared to those of other algorithms. Nevertheless, this method's efficacy could be limited by the system's scalability, especially when handling a high volume of tasks or processors, which could result in more computational complexity and lower efficiency.

An improved technique for cloud task scheduling was proposed by Saravanan et al. [22], termed Improved Wild Horse Optimisation with Levy Flight Algorithm for Task Scheduling in Cloud Computing (IWHOLF-TSC) methodology. To improve resource optimization, including load balancing and energy efficiency, task scheduling in the cloud computing environment was addressed by using a type of symmetry. Using the cloud computing platform's resources to reduce makespan and maximize utilization, the IWHOLF-TSC approach created a multi-objective fitness function. The method improved work scheduling efficiency by combining the Levy Flight (LF) theory and the Wild Horse Optimisation (WHO) algorithm. However, the complexity of real-world cloud settings may affect its performance related to throughput, since task variety and unpredictability may result in less-than-ideal scheduling results.

Mangalampalli et al. [23] suggested to use of the Multi-Objective Trust-Aware Task Scheduling Algorithm (MOTSWAO) scheduler, which assessed task and virtual machine (VM) priority and scheduled tasks in accordance with virtual resources. The Whale Optimisation Algorithm was utilized in the development of this scheduler. The CloudSim toolbox was used to implement the complete complex simulation. One drawback of this strategy was that it would not be as accurate at prioritizing tasks and virtual machines (VMs) in extremely dynamic cloud settings, which results in inefficient use of resources and task execution.

Alsaidy et al. [24] suggested an improved way to initialize Particle Swarm Optimisation (PSO) using heuristic methods. The PSO was initialized using the Longest Job to Fastest Processor (LJFP) and Minimum Completion Time (MCT) algorithms. The suggested LJFP-PSO and MCT-PSO algorithms were assessed for their ability to minimize the metrics of makespan, total execution time, degree of imbalance, and total energy consumption. The first particles produced by the heuristic initialization of the PSO population all began the search from the same beginning point. However, especially in complicated or highly dynamic task settings, relying solely on a single starting point may result in inefficient task prioritization and longer convergence to optimal solutions.

Tuli et al. [25] presented HUNTER, an artificial intelligence (AI)-based comprehensive resource management approach for sustainable cloud computing. The model included three key models—energy, thermal, and cooling—to define the purpose of optimizing energy efficiency in data centers as a multi-objective scheduling issue. To produce the best scheduling choices and approximate the Quality of Service (QoS) for a system state, HUNTER used a Gated Graph Convolution Network as a surrogate model. The Resource Monitor provided resource measurements, while Container Orchestration was used to schedule tasks, which were realized as containers inside the system. However, especially in large-scale or highly changeable clouds, the complicated multi-objective character of the scheduling issue results in longer makespan and slower convergence to optimal solutions.

By combining messy inertia weight based on random assortment with the swarm intelligence of social spiders, Patel et al. [26] aimed to improve load balancing and reduce the total makespan. To avoid local convergence and investigate global intelligence in choosing the most optimal virtual machine (VM) for user activities, the suggested method was created. Nevertheless, this method's drawbacks included possible problems with costs, makespan, and memory use. Because of the algorithm's intricate swarm-based methodology, there could still be an increase in computing cost and a longer makespan. In addition, because swarm intelligence processes are complex, there will be additional costs involved in installing and maintaining the algorithm, and memory use may be less effective.

A multi-objective virtual machine (VM) placement strategy for edge-cloud data centers was presented by Nabavi et al. [27]. Seagull Optimisation is used to simultaneously optimize power consumption and network traffic. By concentrating virtual machine (VM) connections inside the same PMs, this method attempted to decrease network traffic between physical machines (PMs) and therefore the quantity of data moved via the network. Furthermore, it aimed to minimize PM power use by grouping virtual machines into fewer PMs, which resulted in lower energy consumption. Two distinct network topologies, VL2 (Virtual Layer 2) and three-tier were examined through simulations carried out in CloudSim. However, because of possible resource competition and inefficiencies in task scheduling, the focus on condensing VMs to fewer PMs may result in an increased makespan.

There are many drawbacks to the resource allocation and job scheduling approaches that have been examined for cloud computing. Numerous methodologies encounter obstacles associated with deteriorating performance in intricate settings, leading to extended makespan and decreased efficacy [18,19,22]. In extremely dynamic environments, there are additional problems with task prioritization and delayed convergence to optimal solutions [23,25]. Higher costs and inefficiencies in memory use are also frequent issues [24,26]. Another restriction is a longer makespan as a result of resource rivalry, particularly when resources are consolidated into fewer physical machines [27]. These frequent problems highlight challenges with convergence, cost, scalability, and efficiency across different cloud computing resource management techniques.

2.1 Motivation for the Research and Key Goals

There are some drawbacks to using the traditional Salp Swarm Algorithm (SSA) for cloud scheduling issues. Its poor convergence rate is one of the main problems, as it might lead to less-than-ideal solutions, particularly in cloud settings that are constantly changing. This gradual convergence lessens the algorithm's usefulness in practical cloud environments by impeding its capacity to swiftly and effectively adjust to changing jobs or resource availability. Additionally, existing hierarchical models frequently rely on First-In First-Out (FIFO) scheduling methods that fail to consider real-time resource availability, which leads to underutilization of resources and unnecessarily extended makespan times, negatively impacting the overall performance of cloud services. The lack of adaptive scheduling mechanisms prevents optimal allocation and utilization of resources, leading to inefficiencies, particularly in managing job prioritization during periods of high workload. If tasks are not prioritized according to urgency or deadlines, critical jobs may be delayed, leading to missed deadlines and inefficient completion. Finally, another drawback of the conventional SSA strategy is its reliance on a single leader to direct the optimization process. This single-leader approach limits the search space's investigation, frequently resulting in less-than-ideal scheduling solutions. The algorithm could lose out on superior answers that could be identified with a more diversified or multi-leader approach if it just relies on one leader.

Due to the drawbacks of both the current hierarchical cloud scheduling models and the classic Salp Swarm Algorithm (SSA), a unique method is required to address these issues. To improve convergence speed, maximize resource utilization, efficiently prioritize jobs, and broaden the search area for superior scheduling solutions, a new strategy is required. The unique solution will greatly increase the flexibility and efficiency of cloud scheduling by resolving these problems, guaranteeing more reliable performance in dynamic and resource-constrained contexts.

This work combines an enhanced multi-objective Salp Swarm Algorithm with a hierarchical clustering phase to present a novel two-phase scheduling algorithm named Clustered Input Orientated Salp Swarm Algorithm (CIOSSA) for cloud scheduling. This approach seeks to enhance convergence speed, maximize resource utilization, efficiently prioritize jobs, and expand the search space for optimal scheduling solutions. The basic framework followed is hierarchical modeling.

The following are the key goals of CIOSSA:

- CIOSSA improves the sluggish convergence rate and broadens the search space to produce better scheduling solutions by adding two distinct leaders to the Salp Swarm Algorithm.
- For scheduling resources, this method dynamically takes into account the availability of resources at the time, ensuring better resource utilization avoiding overloading or underutilization of resources, and increasing makespan effectiveness.
- By classifying activities according to their due dates, Task Splitting Agglomerative Clustering (TSAC) facilitates more efficient management of urgent jobs and better prioritization, which improves scheduling efficiency in general.

The remaining paper is structured as follows. Section 2 describes the prerequisites; Section 3 presents a comprehensive depiction of the proposed framework. The illustration of the algorithm and mathematical model is included in Section 4. Section 5 carries the result analysis.

2.2 Pre-Requisites

The proposed method follows a hierarchical modeling approach with some modifications. It also uses the hierarchical clustering method for task categorization and a modified SSA for scheduling.

2.3 Hierarchical Modeling Approach

The cloud computing system functions using a master-slave structure for resource allocation and management, according to the hierarchical modeling technique. In this case, the "slave" nodes are in charge of administering the virtual machines inside each cluster, while the "master" node is in charge of controlling resource distribution and user requests. Resource allocation takes place at the virtual machine level, with the VMs hosted on physical machines (PMs). PMs are divided into three states according to their availability and task-handling readiness: warm, cool, and hot. Cold systems have virtual machines that are switched off and take a long time to activate, warm systems have operational VMs but are not yet prepared to process tasks, and hot systems have VMs that are active and can execute processes right away.

In the hierarchical modeling approach, resource allocation and management are maintained master-slave structure. The master manages users' requests and resources. The VMs are managed at a cluster level. The VMs are deployed in Physical Machines. The virtual resource allocation is done at the VM level [4]. The basic diagrammatic representation is shown in Fig. 1. The PMs that are involved in cloud computing services are categorized as hot, warm, and cold. Hot systems are those having active VMs able to run jobs at this moment, warm means VMs are not able to run the job at this moment,

and cold means the VMs are turned off. The cold state requires more time to turn to active mode as compared to warm.

Nevertheless, there are a number of technical issues with the current hierarchical model:

- The existing models do not take into account the various states that VMs may be in since it only takes into account one active PM state at a time. This constraint limits how flexible the model can be when allocating and managing resources, which results in less-than-ideal use of the resources that are available.
- First In First Out (FIFO) scheduling is used by the model, which executes tasks in the order that they are received. This method might result in inefficiencies since it ignores the changing availability of resources. Longer wait times and possible resource bottlenecks arise from FIFO scheduling's inability to adjust to changing resource needs or the status of VMs.
- The model does not check if the necessary resources are currently available when allocating tasks. Deficits in resources are only discovered during execution, which might cause delays and wasteful usage of resources. In the event of unanticipated resource limitations, jobs may be queued up or postponed if proactive resource assessment is lacking.
- If the necessary VMs are distributed among several PMs, the approach may result in inefficiencies since it assigns all tasks inside a job to a single PM. A new task cannot be handled right away and must wait until appropriate VMs are available inside a single PM if the VMs required for execution are dispersed across many PMs rather than being consolidated in one. This restriction makes the pool of available resources less effective and increases delays.

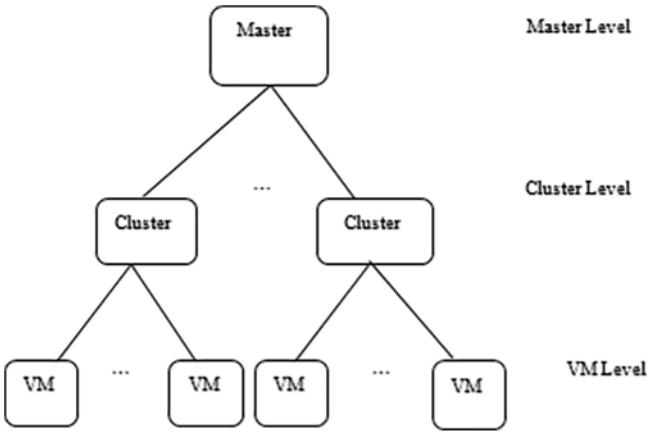


Figure 1: Hierarchical model

2.4 Hierarchical Clustering

In data analysis, hierarchical clustering is a technique that groups data points into clusters according to how similar they are. A dendrogram, which has a tree-like structure, is a visual representation of the hierarchy of clusters created by hierarchical clustering, in contrast to other clustering approaches that call for a certain number of clusters. This approach offers a technique to investigate the natural grouping within the data, which is especially helpful when the link between the data points is not well defined.

As there are as many clusters as there are data points, the method starts by treating each data point as a separate cluster. With this method, the algorithm may progressively advance from the most

granular level—where every data point is a separate entity to bigger clusters, which combine points that are related. Measuring the similarity between every pair of clusters is the next stage. This is usually accomplished using a distance metric, the most often used of which being Euclidean distance. Since it calculates the straight-line distance between two locations in a multidimensional space, the Euclidean distance is a popular choice for determining how similar two pieces of data are. The method finds the two clusters that are closest to one another and combines them into a single cluster after calculating the distances between each pair of clusters. The number of clusters in the data set is decreased by iteratively repeating this process and merging the closest pair of clusters each time.

A dendrogram, a figure that resembles a tree and illustrates the sequence in which clusters are joined, is used to illustrate the process of merging clusters. Clusters at various granularities can be identified thanks to the dendrogram, which offers a visual depiction of the data's hierarchical structure. Up until a termination requirement is satisfied, the merging process goes on. This could happen when all the data points are consolidated into one cluster, or it might depend on a preset similarity threshold, in which case clusters are split up if they get too close to one other. To extract the required number of clusters, the dendrogram can be "cut" at various levels once the hierarchical clustering procedure is finished. One may identify the ideal number of clusters for a particular data set by examining the dendrogram to determine when to stop merging. In hierarchical clustering, comparable data points are clustered together because of the algorithm's ability to measure and compare distances across clusters, which is made possible by the distribution of data points throughout Euclidean space.

2.5 Salp Swarm Algorithm

The Salp Swarm Algorithm (SSA) is an optimization approach inspired by nature that simulates the cooperative behavior of salps in the water, where their shared activity of seeking food forms a salp chain. The leader at the head of the chain directs movement, while the followers change their locations in response to the leader's instructions. The population of potential solutions is comparable to the salp chain in the context of optimization, and the food source (F) is represented by the best solution in the population.

Every solution has many dimensions, where the quantity of dimensions (n) is equivalent to the quantity of variables in the issue. All salps (or solutions) are positioned and their coordinates (x and y) are recorded in a two-dimensional matrix. Successful mapping of this method to scheduling issues is evidence that it is a viable technique Huang and Yeh [13]. The authors evaluated SSA on a variety of mathematical optimization functions. However, SSA's sluggish convergence rate is a major drawback, especially with regard to cloud scheduling. This decreases SSA's capacity to identify optimum solutions fast in dynamic situations and hinders the effectiveness of its search.

3 Problem Description

According to earlier research, the Salp Swarm Algorithm (SSA) has demonstrated potential for resolving cloud scheduling issues. The Clustered Input Orientated Salp Swarm technique (CIOSSA), a resource allocation technique, has been developed to build on the strengths of SSA. The goal of CIOSSA's design is to maximize the scheduling of diverse and independent tasks in cloud settings. When a work is described as "heterogeneous" in this sense, it signifies those various occupations require varied amounts of computational resources. The goal of CIOSSA is to minimize the makespan, or the total amount of time needed to do all activities on the schedule. This reduces operating expenses and maximizes resource utilization. A Physical Machine's (PM) waiting queue is managed by the algorithm, which makes sure that resource allocation is carried out in a way that best distributes

workload among available resources. With this focused approach, resource waste is minimized and job execution efficiency is increased in cloud computing settings, boosting overall system performance.

3.1 Basic Architecture

Resources like RAM, vCPU, and disc space may be accessed whenever needed thanks to cloud computing. These resources are stored on virtual machines that are physically installed within larger computers. Customers can request these cloud computing services from Cloud Service Providers (CSPs), and the effectiveness of the resource allocation process has a major impact on the cost and quality of Infrastructure as a Service (IaaS). To guarantee that client demands are satisfied both effectively and economically, optimal resource allocation is essential. Cloud scheduling's main goal is to allocate resources in an optimal or nearly optimal way to meet user needs.

The customer's request represented as heterogeneous tasks $T = \{T_1, T_2, T_3, \ldots, T_n\}$. These tasks carried out by the allocation of VMs say $VM = \{VM_1, VM_2, VM_3, \ldots, VM_m\}$.

The chief goal of the proposed framework is to create a resource allocation algorithm in the cloud environment. The Clustered Input Orientated Salp Swarm technique (CIOSSA), a resource allocation technique created especially for cloud environments, is introduced in the suggested framework to solve this. The objectives of CIOSSA are to optimize resource utilization, reduce operating expenses, and guarantee that the final resource allocation plan satisfies the intended performance standards. By taking into account many parameters, this algorithm optimizes resource allocation, resulting in a more equitable and economical delivery of cloud services.

The architecture of the proposed method is verified in Fig. 2. It is modeled as the Entry part, Clustering phase, and Optimization phase. The task is entered into the task entry stage. The clustering stage considers the tasks in the entry queue and categorizes them into low-range and high-range tasks. The deadline is the margin for this categorization and is described in Section 4.1.

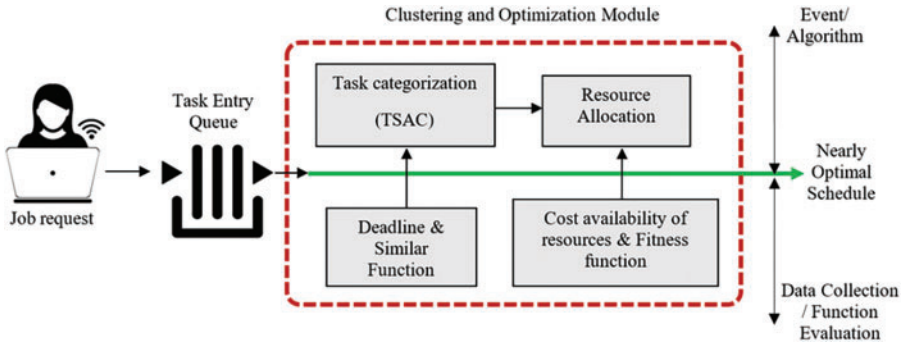


Figure 2: Architecture of CIOSSA framework

Task Splitting Agglomerative Clustering (TSAC), a kind of hierarchical agglomerative clustering, is the clustering technique used in the suggested framework. In contrast to conventional methods, the main goal of TSAC is to order jobs according to their urgency, which successfully overcomes the drawbacks of the First-Come, First-Served (FCFS) rule. TSAC ensures that activities are completed in a way that corresponds with their criticality by implementing a deadline-based prioritization method, which increases overall system efficiency. The second phase of the framework is optimization, which is accomplished by means of an improved Salp Swarm Algorithm (SSA) called the Improved Salp Swarm Algorithm (IOSSA). Swarm-based optimization approach IOSSA ensures more efficient resource

allocation by directly integrating an objective function into the job scheduling process. This method exceeds the effectiveness of current schemes by taking into consideration both resource availability and user requirements when allocating resources. A reliable method for dynamic and effective cloud scheduling is provided by the combination of TSAC for task prioritization and IOSSA for optimal resource allocation.

Section 4.2 explains the classification of tasks from the task entry phase by incorporating clustering. The categorized tasks from the clustering phase are taken as input by the optimization stage that is described in detail in Section 4.3.

3.2 Problem Formulation

The fundamental idea behind the suggested approach is to create an ideal job scheduling algorithm that is suited for cloud systems, with the main goals being to minimize costs and makespan. Makespan is the maximum completion time among all tasks inside a schedule. It is defined as the total amount of time needed from the beginning to the end of executing a group of activities. Therefore, the goal of an ideal task scheduling algorithm is to minimize the makespan in order to guarantee that jobs are finished as quickly as feasible. The algorithm's reduction of the makespan not only improves resource utilization but also lowers operating expenses related to longer job execution durations. This method guarantees the efficient allocation and management of cloud resources, which enhances cloud environment performance and reduces costs.

The main objective of the proposed framework is formulated using Eqs. (1)–(7).

$$Obj\,(S) = \mu XObj_1\,(S) + (1-\mu)\,XObj_2\,(S) \tag{1}$$

where $0 < \mu < 1$.

$$Obj_1\,(S) = min\,(Makespan\,(S)) \tag{2}$$

$$Makespan\,(S) = max \sum_{i=1,2,\ldots m} \left(VM_i\left(CTT_j\right)\right) - min_{i=1,2,..m}\left(VM_i\left(STT_j\right)\right) \tag{3}$$

where CTT_j is the task completion time and STT_j is the start time of task. The variables that are used in this paper are defined in Table 1. The parameters under consideration during the scheduling comprise vCPU cost and requirement of memory and its cost. The cost is computed using Eqs. (4)–(7). The low-cost tasks earn more priority, which may cause missing of the deadline by some other tasks. In the proposed method, a task classification is introduced in terms of deadline, to cope with that deadline constraint.

$$Obj_2\,(S) = min\,(Total_{cost}) \tag{4}$$

$$Total_{cost} = f\,(TotvCPU_{cost}TotMem_{cost}) \tag{5}$$

$$TotvCPU_{cost} = (ReqvCPU * vCPU_{cost}) \tag{6}$$

$$TotMem_{cost} = (Req_{Mem} * Mem_{cost}) \tag{7}$$

A thorough description of the variables utilized in the suggested job scheduling algorithm in the cloud environment can be found in Table 1.

Table 1: Variable description

Variables	Description
T_iT_j	Cluster of tasks
Dt_x	Deadline of tasks t_x in T_i
Dt_y	Deadline of tasks t_y in T_j
$Total_{cost}$	Total cost
$vCPU$	Need of $vCPU$
Mem_{need}	Need memory
$TotvCPU_{cost}$, $TotMem_{cost}$	Total $vCPU$ and Memory cost
$vCPU_{cost}$, Mem_{cost}	$vCPU$ and Memory cost
Req_{Mem}, $ReqvCPU$	Required $vCPU$ and Memory
KC	TEQ waiting buffer size
TEQ_{front} & TEQ_{rear}	Points to front and rear ends of Task Entry Queue (TEQ)
t	Current iteration
lb[ ]	Lower bound
ub[ ]	Upper bound
X_j	Position of salp
F_j	Position of food source
N	Population/number of tasks
I_{best}	The best solution for each iteration
N_{req}	Maximum iterations/total number of requests

These variables described in Table 1 include a wide range of topics related to resource allocation, such as cost estimations, deadlines, job grouping, and resource needs. Comprehending these factors is essential for maximizing makespan, guaranteeing effective resource usage, and optimizing job scheduling.

4 Proposed Model

Task Entry, Clustering, and Optimization are the three main phases of the suggested model's organization. At first, the framework introduces a more dynamic method that ranks tasks according to resource availability and urgency, changing the conventional FIFO task selection procedure. The model dynamically verifies the resources' availability during the customer resource allocation phase, making sure that the allocation is effective and sensitive to the circumstances of the moment. A multi-leader Salp Swarm Algorithm (SSA) is integrated into the model during the Optimization stage to improve the search process and enable a wider investigation of possible solutions. By expanding the search area and raising the possibility of finding a close to ideal solution, the multi-leader strategy greatly enhances the algorithm's capacity to identify an ideal timetable and overcomes the drawbacks of single-leader techniques. These three phases are explained in the upcoming sections.

4.1 Task Entry

The Task Entry Queue (TEQ) is a crucial control mechanism in the Task Entry segment of the proposed model that handles incoming user requests. Upon submission of a job by a user, which usually entails requesting certain cloud resources, the system assesses the TEQ's current load by first evaluating its status. Because the TEQ is built with a limited capacity to prevent the system from being overloaded by too many simultaneous requests, this evaluation is crucial. The TEQ instantly discards the newly entered task if it is full, or has reached its maximum capacity. To keep the system stable and avoid any overloads that can cause delays, resource conflicts, or even system failures, this drop is an essential step. The system makes sure that only a manageable number of tasks are handled, which aids in maintaining ideal performance levels, by rejecting jobs when the queue is full.

In contrast, the newly submitted task is added to the queue if the TEQ has not yet reached its capacity. In order to guarantee that jobs are completed in a methodical manner and enable the system to handle each task effectively, this modification is made. Maintaining the cloud scheduling system's overall throughput and performance depends on the organized administration of jobs within the TEQ.

The pseudo-code representation of the task entry-stage is depicted in Algorithm 1.

Algorithm 1: Task entry

Input: Enter job into TEQ with maximum buffer size in KC
Output: Allot job in the queue
TEQ is the array representation of Queue structure
with TEQ_{front} and TEQ_{rear} points front and rear ends.
Algorithm Task Entry (Job)
{
$TEQ_{front} = 0$, $TEQ_{rear} = -1$;
While $\left(TEQ_{front} \leq KC - 1\right)$
$TEQ_{rear} + +$;
$KC + +$;
Enqueue (Job)
End while
TEQ is full and Drop Job.
}

Hence, to maintain system load balance and guarantee regulated and effective resource allocation, the Task Entry segment's task management strategy is crucial. This procedure is essential for preserving the scheduling system's stability and efficiency since it avoids bottlenecks and guarantees that the system runs within its intended capacity and it helps the later stages of the model run smoothly by keeping the system from being overloaded. When a job is entered, the system moves on to the Clustering step, where it is grouped for further optimization according to its priority and resource needs.

4.2 Task Splitting Agglomerative Clustering (TSAC)

By utilizing the concepts of hierarchical clustering, the work Splitting Agglomerative Clustering (TSAC) system presents a unique approach to work prioritization and scheduling in cloud settings. By classifying jobs into discrete priority levels according to their due dates, TSAC modifies the

conventional FIFO scheduling approach. Before tasks are allocated to high-priority or low-priority categories in this system, they are first reviewed.

Consider there are 10 different tasks T_1, $T_2 \ldots T_{10}$ is in the Task Entry Queue and they are entered in the order of T_1 first, T_2 second, and so on. Their deadlines are 93, 78, 10, 67, 81, 89, 21, 34, 9, 26 ms, respectively. Initially, tasks are split into two categories high-priority and low-priority based on their deadlines. Tasks with shorter deadlines are categorized as high-priority, while those with longer deadlines are placed in the low-priority category. The tasks having deadlines 93, 78, 67, 81, and 89 say T_1, T_2, T_4, T_5, T_6 are in low priority category and tasks having deadlines 10, 21, 34, 9, and 26 that is T_3, T_7, T_8, T_9, T_{10} are in high priority category. Suppose the tasks in category one are selected for scheduling T_1, T_2, T_4, T_5, T_6 are scheduled first. This changes the default scheduling policy FIFO of the Hierarchical Modeling Approach.

In this scheduling approach, high-priority tasks, having shorter deadlines, are scheduled before low-priority ones. This prioritization mechanism ensures that the tasks with urgent deadlines will be executed as soon as possible to minimize the possibility of missing their deadline and further enhance the overall efficiency in scheduling. Thus, TSAC modifies the default FIFO, which performs tasks strictly in arrival order, to apply a priority basis according to deadlines for a more accurate reflection of both task urgency and system requests. The proposed model will implement the TSAC to handle such drawbacks compared to traditional scheduling methods, many of which have inefficiencies in dealing with urgent tasks and poorly utilize resources.

TSAC uses a hierarchical clustering method to group tasks. Each task is initially treated as a separate cluster, and clusters are merged based on their similarity, specifically their deadline proximity of the tasks. Distance between clusters T_i and T_j is the minimum distance between any object in T_i and T_j. The distance measure is based on the deadline of tasks. The mathematical model for similarity calculation is using Eq. (8). Merge clusters based on maximum similarity.

$$Sim\left(T_i,\ T_j\right) = Max\ sim\left(Dt_x,\ Dt_y\right) \tag{8}$$

where $t_x \in T_i$ and $t_y \in T_j$.

The goal is to merge clusters that have the most similar deadlines until there are only two clusters left: one for high-priority tasks and one for low-priority tasks. The algorithm iteratively merges the two closest clusters based on the calculated similarity until only two clusters remain (high and low priority). Once tasks are clustered, TSAC adjusts the conventional FIFO scheduling approach by giving high-priority tasks precedence over low-priority ones. High-priority tasks are executed first to prevent missing deadlines, ensuring that tasks with urgent deadlines complete on time. TSAC dynamically prioritizes the tasks according to their deadlines, higher performance and lower makespan will result due to more effective resource usage. The clustering algorithm is shown in Algorithm 2.

Algorithm 2: Task splitting agglomerative clustering (TSAC)

Input: Tasks to be scheduled.
Output: Two clusters, high range and low range
Initially each task makes a cluster search space.
Set CL as total number of clusters.
Evaluate $Max\ sim\left(Dt_x,\ Dt_y\right)$ using Eq. (8)
While ($CL \leq 2$):
Merge the two closest clusters

(Continued)

Algorithm 2 (continued)

Update $Sim\left(T_i,\ T_j\right)$
Decrement CL by 1
End while

Following this, the framework progresses to the Optimization stage, where the improved Salp Swarm Algorithm (IOSSA) is applied. IOSSA further refines the task scheduling process by optimizing the resource allocation based on the prioritized task categories. This optimization aims to enhance the efficiency of resource utilization and minimize overall scheduling costs, thereby addressing the limitations of previous approaches and achieving a more effective scheduling solution.

4.3 Input Oriented Salp Swarm Algorithm (IOSSA)

In the proposed framework, a variation of the SSA named IOSSA is utilized for the optimization process. Traditional SSA employs a single leader for goal searching, which leads to reduced performance and exploration capability. In contrast, IOSSA introduces two distinct leaders, each addressing separate objectives. This dual-leader approach significantly expands the search space, allowing for more comprehensive exploration of potential solutions.

The introduction of dual leaders not only improves the algorithm's search capabilities but also facilitates better convergence towards optimal solutions. Each leader focuses on different aspects of the scheduling problem, thus reducing the risk of premature convergence often seen in single-leader algorithms. This strategic enhancement is particularly beneficial in solving complex scheduling issues, where multiple conflicting objectives exist. By balancing the exploration and exploitation phases across two leaders, IOSSA adaptively navigate the solution space, ensuring that critical tasks are prioritized without compromising overall resource efficiency.

In general, the SSA will have initial population settings depending on a random selection procedure, where it has several shortcomings when dealing with large and complex sets of tasks within a cloud computing environment. As opposed to this, there is more strategical task scheduling involved within the proposed framework using the algorithm known as Task Splitting Agglomerative Clustering. On the other hand, instead of randomly selecting tasks, TSAC categorizes the tasks based on priority. This process ensures that tasks with urgent deadlines will be given more priority. The goal is to align resource allocation better with customer requirements and reduce the chances of resource bottlenecks. After categorizing these tasks, this algorithm checks the current availability of resources within the resource pool. If the pool can provide sufficient resources, then those resources are assigned to high-priority tasks. When the resources in the pool are not enough to meet the demands of prioritized tasks, the tasks would be distributed intelligently by the system across the available VMs for maximum utilization of resources. A significant advantage of our approach is the dynamic checking of resource availability before task allocation. By assessing the current state of resources such as CPU and memory availability before assigning tasks, the algorithm ensures optimal resource utilization. This allocates resources based on real-time availability rather than relying on static assumptions or previous allocations. This means that if certain resources are not available, tasks are intelligently redistributed across the cloud infrastructure. This approach leads to more balanced resource utilization, minimizes waiting times for tasks, and helps avoid situations where tasks are queued due to insufficient resources. Dynamic resource allocation strategy directly impacts cost efficiency. This adaptive resource management aligns closely with the objectives of cloud computing delivering efficient, scalable, and cost-effective services to users. This approach, besides improving the

efficiency of SSA by reducing inefficiencies due to random allocation, transforms critical tasks into timely executions to optimize the overall performance of the cloud environment.

This stratification and resource allocation strategy in TSAC ensures that the cloud infrastructure will be able to adapt dynamically to varying workloads, hence making the management of cloud resources more effective and efficient. During the optimization step, this would be further refined by using an improved SSA for achieving optimal scheduling outcomes. The proposed IOSSA is shown in Algorithm 3.

Algorithm 3: Input oriented salp swarm optimization (IOSSA)

Input: Tasks to be scheduled
Output: Nearly optimal schedule
Initialize the population based on number of tasks n.
Initialize maximum iterations based on total number of requests.
Set upper bound (ub_1 & ub_2) and lower bound (lb_1 & lb_2).
Initialize (C_1, C_2 *and* C_3) using random function.
Set the initial iteration $t = 1$.
Generate the initial populations.
1. Categorize the tasks using Task Splitting Agglomerative Clustering (TSAC) Algorithm.
2. Availability of resource is checked and select task category.
3. Select the task category bsed on TSAC.
4. Identify individual items in population ($m!/2$).

While ($t < max\,(t)$)
Evaluate two fitness values for each schedule using fitness function Obj_1 and Obj_2.
Identify the best solution F_1 for Obj_1 and Obj_2.
Set k as 1 to handle each objects in intial population.
Update C_1 according to Eq. (10).
$k = 1$
for no item in the poulaion remains to visit
if ($k == 1$)
Update the position of $leader_1$ and $leader_2$ using Eq. (9).
else
Update the position of follower of Obj_1 and Obj_2 using Eq. (11).
End for
Increment t by 1
Update lower and upper limits.
End while
Identify the best out of two using Eq. (1).
Return the best solution

The proposed technique involves one of the major overheads in the initial phases: the strategic selection of task categories, playing a vital role in the optimization of resource utilization. Categorization depends upon key resources such as vCPU, memory, and associated cost. By analyzing these parameters, the algorithm intelligently decides to either select high-range tasks or low-range tasks for scheduling. This decision-making will play a crucial role in not allowing resource overloading. Thus, when the resource availability is high, the system concentrates on high-range tasks. Although these generally require higher resources, they give a high contribution in workload completion. On

the contrary, when the VMs are busy or resource availability is low, then the algorithm targets low-range tasks. The reason is that low-range tasks have a shorter deadline and can execute faster, hence not delaying the important tasks. It ensures that this prioritization mechanism makes sure that the most important tasks will be given the attention and resources they need, hence allowing the cloud environment to be even more efficient.

The scheduling itself is a two-tier process: resource allocation and actual task scheduling. Resource allocation includes assigning VMs to the selected tasks, while in task scheduling, time slots are allocated for these tasks in the chosen VM. Such a two-tier process will achieve an optimal alignment of both resources and timelines of task execution with minimum delay and maximum throughput. A key consideration in the Salp Swarm Algorithm (SSA) is the determination of the number of iterations, which directly impacts the convergence and efficiency of the scheduling process. In the improved SSA (IOSSA) used in this method, the number of iterations is dynamically adjusted based on the task count. It follows from here that with the increase in the number of tasks, the complexity increases along with the potential combinations of scheduling, which demands more iterations. The two-step resource allocation process further enhances scalability by allowing the algorithm to adaptively reallocate resources based on real-time demand. This adaptability ensures that even as workloads expand, the algorithm continues to provide optimal resource utilization and scheduling efficiency without significant degradation in performance and quickly focus on promising solutions. As a result, IOSSA converges more quickly to optimal resource allocations without unnecessary computational overhead, facilitating faster decision-making and reducing overall scheduling times. This adaptive iteration strategy ensures that the scheduling process is robust and capable of finding near-optimal solutions even with increased scaling in the workload for better overall effectiveness of the proposed framework.

The makespan and scheduled total cost is thus minimized by the proposed IOSSA. The follower tends to change their location according to a sole leaders salp in existing SSA that causes performance reduction. Therefore, the proposed IOSSA utilizes two salp chains having separate leaders. Among them one salp chain, i.e., $Obj_1\,(S)$ takes care of the makespan and the other salp chain $Obj_2\,(S)$ reduces the total cost. From these two salp chains, the best solution is constructed by Eq. (1). Though SSA is employed to solve complex optimization issues in some cases sub-optimal solution is obtained due to a lack of global searching ability. However, due to two salp chains utilization, this issue is resolved.

4.4 Mathematical Model for IOSSA

For the mathematical representation, each object in a population that is the schedules is divided into two groups: leader and followers. The leader changes their position based on the requirement of CPU and memory. The leader updates the position using Eq. (9).

$$Sol\,[1] = \begin{cases} F_k + C_1\left(\left(ub\,[k] - lb\,[k]\right) C_2 + lb\,[k]\right) C_3 \geq 0.5 \\ F_k - C_1\left(\left(ub\,[k] - lb\,[k]\right) C_2 + lb\,[k]\right) C_3 < 0.5 \end{cases} \tag{9}$$

where $Sol\,[1]$ represents the position of leader salp, F_k is the food position, $ub\,[k]$ indicates the kth dimension upper bound and $lb\,[k]$ is kth dimension lower bound; C_1, C_2 and C_3 are coefficients and are random numbers. The parameter C_1 handles exploration and exploitation and is calculated using Eq. (10). Where t is the current iteration as well as $max\,(t)$ is the maximum iteration count. C_2 also C_3 are a random number in the range [0, 1].

$$C_1 = 2e^{-(4t/max(t))} \tag{10}$$

By maintaining a balance between global exploration and local exploitation, the two-leader mechanism improves the algorithm's ability to adapt to changes in problem parameters, yielding more robust and reliable solutions. At first, the leader position is updated using Eq. (9) and according to the position of the leader, the followers are updated using Eq. (11).

$$Sol[k] = \frac{1}{2}(Sol[k] + Sol[k-1]) \tag{11}$$

where $t \geq 2$ and $Sol[k]$ indicates the position of kth follower.

An algorithm for CIOSSA is given in the following Algorithm 4.

Algorithm 4: Clustered input oriented salp swarm algorithm

Input:

Set of tasks $T = \{T_1, T_2, \ldots, T_n\}\}$, number of iterations *maxIter*, task priorities, resources available VMs, upper bounds ub_1, ub_2, lower bounds lb_1, lb_2

Step 1: Task Entry Phase:

Initialize the Task Entry Queue (TEQ) with tasks $\{T_1, T_2, \ldots, T_n\}\}$,

Example Input:

T_1 = {deadline:10; vCPU:2; memory:1 GB}

T_2 = {deadline:5; vCPU:1; memory:0.5 GB}

T_3 = {deadline:15; vCPU:2; memory:1 GB}

Check for available capacity in the TEQ

If the queue is full

Incoming task T_4 rejected due to full queue.

If space is available

Enqueue the task T_1 for scheduling.

Step 2: Task Splitting Agglomerative Clustering (TSAC) Phase:

For each task T_i, determine the deadline and resource requirement.

Split tasks into high-priority and low-priority categories based on their deadlines.

Example Input:

- Deadlines: T_1: 10, T_2: 5, T_3: 15
- High-Priority Tasks: T_2
- Low-Priority Tasks: T_1, T_3

Set CL as total number of clusters.

Evaluate $Max\,sim\left(Dt_x, Dt_y\right)$ using Eq. (8)

Merge clusters iteratively until $CL \leq 2$:

Merged clusters of $\boldsymbol{T_1}$ and $\boldsymbol{T_3}$ into low-priority cluster

Update $Sim\left(T_i, T_j\right)$

Decrement CL by 1

Schedule high-priority tasks first.

Output: Scheduled T_2 first.

Step 3: Resource Availability Check

For each task in the high-priority cluster

Check the availability of required resources (vCPU, memory).

Allocate resources dynamically based on current availability.

Example Input: T_2 requires 1 vCPU,0.5 GB memory

If resources are sufficient

(Continued)

Algorithm 4 (continued)

Output: Allocated resources to T_2

If resources are insufficient:

Output: Reschedule low-priority tasks to free up resources.

Step 4: Input Oriented Salp Swarm Optimization (IOSSA)

Initialize coefficients C_1, C_2, and C_3 randomly.

C1 = 0.8

C2 = 1.2

C3 = 0.9

While ($t < maxIter\,(t)$)

Assign two leaders: Leader 1 optimizes makespan, Leader 2 optimizes cost.

For each salp (solution), evaluate two objective functions

Objective 1: Minimize makespan

Objective 2: Minimize total cost

Identify the best solution F_1 for Obj_1 and Obj_2.

Set k as 1 to handle each objects in intial population

Update C_1 according to Eq. (10).

$k = 1$

for no item in the poulaion remains to visit

if ($k == 1$)

Update the position of $leader_1$ and $leader_2$ using Eq. (9).

Output Example:

- Updated position of Leader 1: New Position= [2.5, 0.7]
- Updated position of Leader 2 = [1.5, 0.7]

else

Update the position of follower of Obj_1 and Obj_2 using Eq. (11)

Output Example: Follower updated position based on Leader 2.

End for

Increment t by 1

Update lower and upper limits

End while

Use Eq. (1) to select the best overall solution based on weighted objectives.

Return the best solution

Output: Best solution identified with makespan = 12, total cost = 300.

Output: Optimal resource allocation and task scheduling with minimized makespan and cost

5 Results and Discussion

The evaluation matrices considered in this section include makespan, resource utilization, cost, throughput, convergence speed and memory usage. Makespan is calculated using Eq. (12), and is related to the task start time (i) and task completion time (j). Resource exploitation is the handling of sharable resources like memory and vCPU. It is determined by Eq. (13). Superior resource utilization makes sure that resource idle time is less. The cost value is computed in terms of vCPU and memory needed as in Eq. (14).

$$Makespan = \sum_{j=0}^{m} \sum_{i=0}^{n} (Completion\ time - Start\ time) \tag{12}$$

$$Utilization = Used\ time / Available\ time \tag{13}$$

$$cost = vCPU_{Need} + Mem_{Need} \tag{14}$$

Now, during the result analysis phase, cost evaluation results of CIOSSA are collected and a graph is plotted. In this case, tasks and VMs are heterogeneous. The proposed CIOSSA is compared with SSA regarding makespan, resource utilization, cost, throughput, convergence speed and memory utilization. A variety of tasks are utilized in the assessment of the cost function of CIOSSA: The value of cost is provided by factoring resources, with individual cost and dynamic resource availability.

5.1 Experimental Setup

The simulation result analysis is done on HP PC with intel CORE i5 8th Gen x64 based processor having 1.60–1.80 GHz processing speed and 8 GB of RAM. The proposed framework performance is assessed by the CloudSim simulator. The CloudSim 4.0 toolkit is run on Windows 10 platform NetBeans IDE 8.2 as IDE and it is associated with jdk 1.8.0_111 jdk package.

In the proposed model, the capabilities of VMs such as available memory, the processing speed of VCPU are variant. To evaluate Clustered IOSSA (CIOSSA) several tasks are managed with various sets of tasks and VMs. Here the tasks and VMs are heterogeneous.

5.2 Evaluation of Performance and Comparison Results with Varying Number of Task Counts

5.2.1 Makespan

The makespan (measured in seconds) for several optimization algorithms (PSO, GWO, GA, SSA, Clustered GWO, and CIOSSA) for various task counts (5, 100, 200, 500, and 1000 tasks) is shown in Table 2 and Fig. 3. The makespan was defined as the total time required to complete all tasks in the scheduling environment. The total time taken to complete a set of tasks was measured in seconds. The average makespan across multiple simulations was calculated to assess the algorithm's efficiency in scheduling tasks. The CIOSSA is shown to produce improved makespan.

Table 2: Comparison of makespan of the proposed CGWO and CIOSSA

Number of tasks	Makespan (sec)					
	PSO	GWO	GA	SSA	Clustered GWO	CIOSSA
5	24.05	22.11	20.12	19.35	20.54	18.12
100	54.31	53.35	52.31	51.87	54.67	50.55
200	130.12	120.24	110.42	114.33	115.53	110.23
500	419.33	410.38	388.63	385.56	386.73	378.67
1000	785.42	774.46	761.54	755.43	750.86	744.48

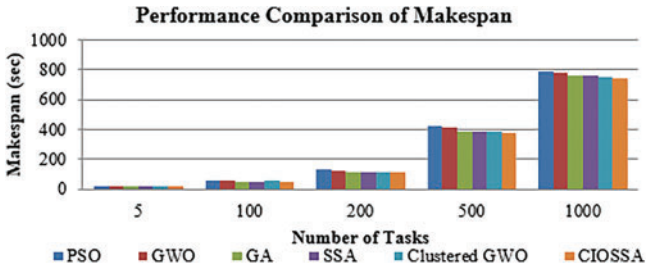


Figure 3: Comparison of makespan of the proposed CIOSSA model over other traditional models and CGWO

The makespan for the proposed CGWO and CIOSSA, GA, PSO, SSA, and Standard GWO is shown in Table 3, along with the Performance Improvement Rate (PIR%). The PIR of resource use demonstrates that the recommended clustered GWO yields 2.02% Clustered GWO. It has been noted that CIOSSA better makespan.

Table 3: Performance improvement rate of makespan with CGWO and CIOSSA

Performance improvement rate in %						
	PSO	GWO	GA	SSA	CGWO	CIOSSA
Total makespan	1413.23	1380.54	1333.02	1326.54	1328.33	1302.05
PIR % over PSO		2.368	6.017	6.535	6.391	8.539
PIR % over Std. GWO			3.565	4.071	3.93	6.028
PIR % over GA				0.488	0.353	2.379
PIR % over SSA					0.135	1.881
PIR % over CGWO						2.02

5.2.2 Resource Utilization

To evaluate which algorithm uses resources most efficiently for various amounts of jobs, the resource consumption statistics can be understood from Table 4. This metric was determined by analyzing the percentage of allocated resources (e.g., vCPU and memory) used during task execution. Higher resource utilization indicates better performance, as it signifies effective resource allocation and minimization of idle time. Each column represents a distinct optimization technique, and each row represents a different number of jobs. The efficiency of resource allocation was then compared across the different algorithms.

Table 4: Resource utilization of proposed CGWO and CIOSSA

Number of tasks	Resource utilization (%)					
	GA	Standard GWO	PSO	SSA	Clustered GWO	CIOSSA
5	63.64	66.51	70.56	72.86	72.16	74.61
100	69.38	72.32	75.55	77.75	72.75	79.95

(Continued)

Table 4 (continued)

Number of tasks	Resource utilization (%)					
	GA	Standard GWO	PSO	SSA	Clustered GWO	CIOSSA
200	71.27	74.67	77.71	79.79	80.76	82.34
500	76.56	80.54	82.66	82.74	82.44	84.94
1000	79.76	80.56	82.81	84.83	84.83	87.55
Mean	72.122	74.92	77.858	79.594	78.588	81.878
Standard deviation	5.6	5.309	4.608	4.15	5.175	4.43
Error margin	±2.5156	±2.37	±2.06	±1.86	±2.314	±1.98

Fig. 4 shows the percentages of resources used by each algorithm for the specified number of tasks. For example, when there are 5 tasks, GA achieves a resource utilization of 63.64%. Standard GWO achieves a resource utilization of 66.51%. PSO achieves a resource utilization of 70.56%.SSA achieves a resource utilization of 72.86%. Clustered GWO achieves a resource utilization of 72.16%. CIOSSA achieves a resource utilization of 74.61%.

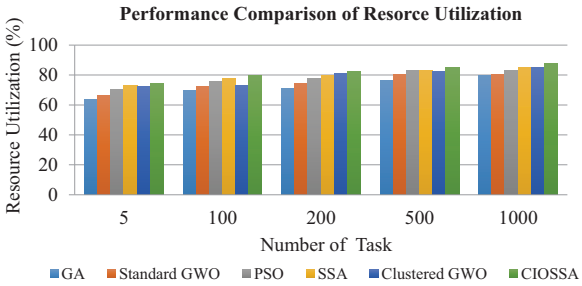


Figure 4: Comparison of resource utilization of the proposed CIOSSA model over other traditional models and CGWO

Table 5 displays the Performance Improvement Rate (PIR%) for the proposed CGWO and CIOSSA, GA, PSO, SSA, and Standard GWO's resource use. The suggested clustered CIOSSA provides 4.018% Clustered GWO, as shown by the PIR of resource usage. It has been highlighted that CIOSSA makes better use of its resources.

Table 5: Performance improvement rate of resource utilization with CGWO and CIOSSA

	Performance improvement rate in %					
	GA	Standard GWO	PSO	SSA	CGWO	CIOSSA
Total	360.61	374.6	389.29	397.97	392.94	409.39
PIR % over GA		3.735	7.367	9.388	8.228	11.915

(Continued)

Table 5 (continued)

Performance improvement rate in %				
PIR % over Std. GWO	3.774	5.872	4.667	8.498
PIR % over PSO		2.181	0.929	4.91
PIR % over SSA			1.28	2.79
PIR % over CGWO				4.018

5.2.3 Cost

The cost function of the CIOSSA assesses utilizing several tasks. Memory and virtual CPU consumption form the basis of the cost value. This approach leads to more balanced resource utilization and helps avoid situations where tasks are queued due to insufficient resources. Dynamic resource allocation strategy directly impacts cost efficiency. The total operational cost associated with resource utilization was calculated based on the dynamic availability and individual costs of resources. Each row in Table 6 represents a different number of jobs, and the accompanying column displays the dollar amount that each task.

Table 6: Cost of CIOSSA

Number of tasks	Cost ($)
50	325.25
80	495.46
100	533.54
200	461.31
300	316.47
500	199.87
1000	191.78

Fig. 5 shows the relation between the number of tasks and the cost within a cloud computing environment. The cost increases with the increase in the number of tasks to a maximum of 100 tasks at $533.54, then decreases drastically to $191.78 at 1000 tasks. This trend indicates the proposed resource allocation and task scheduling approach scales well with an increased task load in terms of cost management and optimization, hence showcasing scalability and cost efficiency for handling a larger workload.

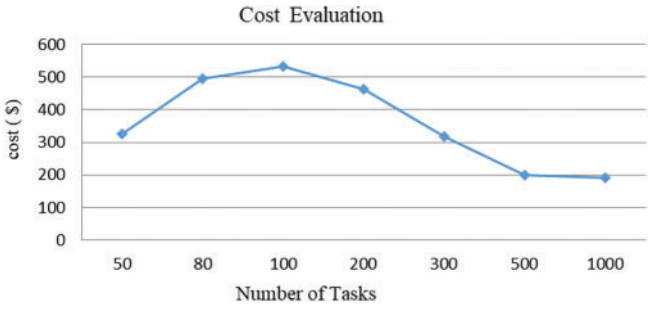
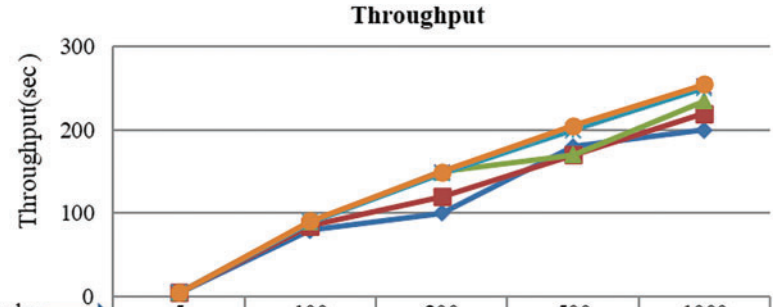


Figure 5: Cost evaluation of the proposed model

5.2.4 Throughput

The number of tasks completed within a given time frame served as a measure of throughput. By monitoring the task completion rate, we were able to assess the efficiency of the proposed algorithm in managing high workloads. The total number of successfully executed tasks within the simulation duration was measured and compared across the proposed approach and baseline algorithms. The comparison of throughput of the proposed CIOSSA over other traditional models like SSA, Standard GWO, PSO, SSA, GA, and Clustered GWO is shown below.

According to the presented data in Fig. 6, the Clustered Input Oriented SSA method obtains the maximum throughput value across various number of tasks. Fig. 6 displays the results of different heuristic methods in terms of throughput. For instance, when there are 1000 jobs to schedule, GA, GWO, PSO, SSA CGWO, and CIOSSA have throughput values of 200, 220, 235, 252, 250, and 255, respectively.

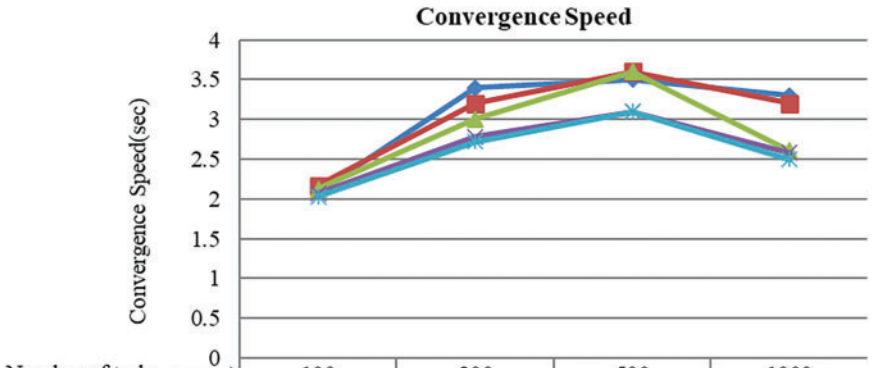


Number of Tasks	5	100	200	500	1000
GA	5	80	100	180	200
GWO	5	85	120	170	220
PSO	5	90	150	170	235
SSA	5	90	150	200	252
Clustered GWO	5	90	148	200	250
CIOSSA	5	91	150	205	255

Figure 6: Comparison of throughput of the proposed CIOSSA model over other traditional models like SSA, Standard GWO, PSO, and GA

5.2.5 Convergence Speed

Convergence speed is the rate at which a scheduling algorithm finds an optimal or nearly ideal solution. Faster convergence indicates an efficient algorithm capable of adapting quickly to changes in resource availability and task priorities. The comparison of the convergence rates of several algorithms, including GA, PSO, GWO, SSA, Clustered GWO, and CIOSSA at various problem sizes is shown in Fig. 7.



Number of tasks	100	200	500	1000
GA	2.09	3.4	3.5	3.3
PSO	2.17	3.2	3.6	3.2
GWO	2.13	3.01	3.6	2.61
SSA	2.07	2.78	3.1	2.58
Clustered GWO	2.03	2.72	3.1	2.5

Figure 7: Comparison of convergence speed of the proposed CGWO model over other traditional models like SSA, Standard GWO, PSO, and GA

From this figure, GA shows the lowest convergence speed number for problems of size 100, indicating a faster convergence rate than the other algorithms. GWO shows the maximum convergence speed value for problem sizes 200 and 1000, indicating slower convergence than the other algorithms. All algorithms have relatively low values for convergence speed for problems of size 1000.

5.3 Evaluation of Performance and Comparison Results with Varying Number of Resources and Tasks

In this section, the performance is evaluated for varying number of resources from 5 to 30 (5, 10, 15, 20, 25, 30) and tasks with 15 and 30.

5.3.1 Cost Evaluation

The cost function of CIOSSA is measured against varying a number of tasks. The cost value is calculated on the dynamic resource availability and the individual cost of resources. It fluctuates in nature and is tabulated in Table 7. The simulation environment has 30 Physical Machines, 86 VMs distributed over this PMs and a total of 166 vCPUs.

As per the cost analysis, if the number of tasks is less than 100, more resources are present, and the idle time of resources increases, thereby boosting the cost value. As the number of jobs grows and we need to fit them into existing resources, there appears to be a higher requirement for task distribution across different VMs. As the number of tasks increases, the cost value will decrease.

Table 7: Cost evaluation

Number of tasks	Cost ($)	Number of tasks	Cost ($)
50	325	200	461
80	495	300	316
100	533	400	206

5.3.2 Makespan

The simulation environment has 30 Physical Machines, 86 VMs distributed over these PMs and a total of 166 vCPUs. One of the widely used software development systems introduced recently is a micro-service system. It comes up with a two-layer resource structure having container or OS-level and VM level.

Alsaidy et al. [24] introduced a GWO in the containerized cloud. The proposed system results are compared with this newly arrived elastic scheduling micro-service system and it is shown in Fig. 8. The results are taken by the assumption that there are 50 containers or 50 active VMs. The proposed system and the two-layer system give almost similar results for makespan.

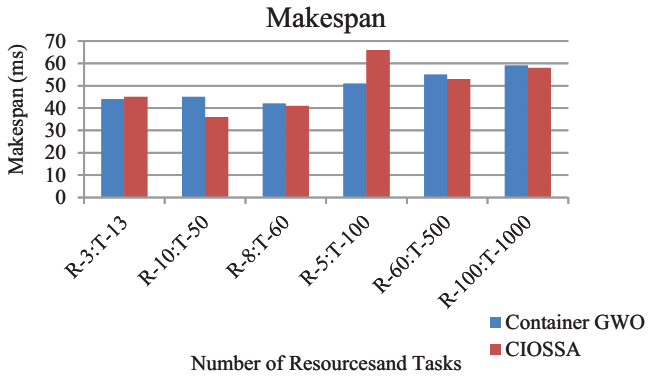


Figure 8: Makespan comparison

Fig. 9 depicts the performance evaluation of the GA, PSO [10], GWO [9], SSA [11], and proposed algorithm CIOSSA in terms of makespan for 85 VMs distributed among 30 PMs. If there are 13 tasks and 3 resources available, the GA, PSO, GWO, SSA, and CIOSSA makespan values are 24.05, 22.11, 20.12, 19.35, and 18.12. When the tasks count is 30 and the resources count is 30, the makespan values are 785.42, 774.46, 761.54, 755.43, and 744.48. Hence it is observed that the makespan of CIOSSA is lower as compared to other existing algorithms. Thus, the resource utilization is increased automatically as the reduction in makespan is observed. The proposed framework produces a 3% improvement in makespan as compared to SSA. It is also noticed that as the number of available resources increased, the makespan was reduced.

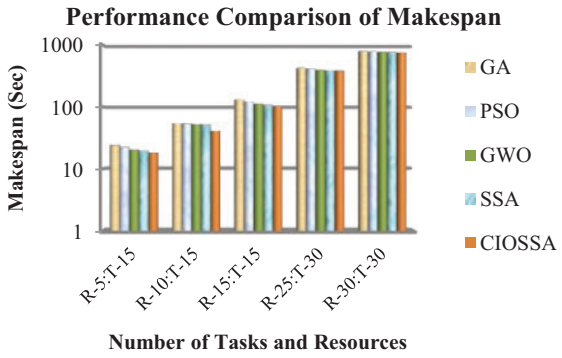


Figure 9: Makespan comparison of different algorithms

5.3.3 Resource Utilization

Fig. 10 depicts the resource utilization comparison. The percentage utilization is marked in the *y*-axis. It is thus observed around a 3% improvement in resource utilization in the proposed framework is observed.

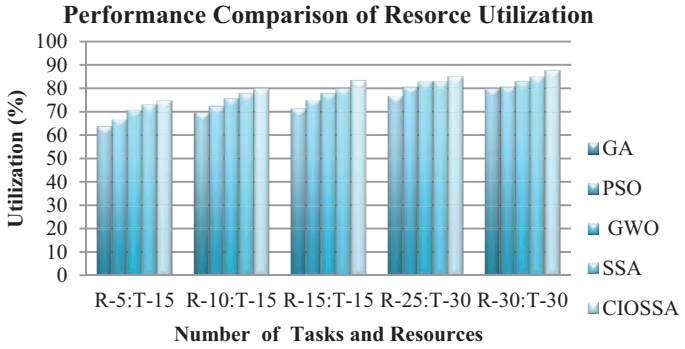


Figure 10: Comparison of resource utilization

5.3.4 Cost and Memory Usage

The cost and memory usage of GA, PSO, GWO, SSA, and CIOSSA are depicted in Figs. 11 and 12. The cost value is based on the dynamic availability of resources as well as their individual costs. 564, 624, 653, 574 and 551 are the total costs observed for various algorithms and are shown in Fig. 11. According to the results of the experiment, GA, SSA, and CIOSSA all had improved cost results, with CIOSSA being the best.

Table 8 exposes a cost and memory usage comparison and here *R* refers to the resources count and *T* refers to the task count. The cost and memory are also noticed for increasing tasks and resources for the proposed algorithm and with the existing SSA algorithm. The memory is measured in Megabytes (MB). This proves that the proposed algorithm offers better resources with low memory utilization. This thus states the efficiency of the proposed algorithm.

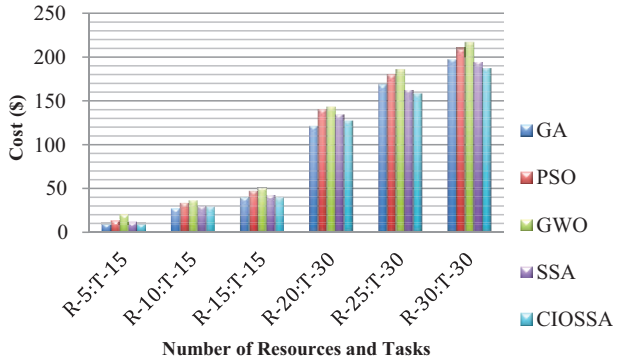


Figure 11: Comparison of cost

Table 8: Comparison for cost and memory

Resources and tasks	Cost ($)		Memory (MB)	
	SSA	CIOSSA	SSA	CIOSSA
R-5:T-15	12.6	9.38	100555	96445
R-10:T-15	30.3	12.4	112545	100542
R-15:T-15	42.3	21.7	135486	114493
R-20:T-30	134	87.3	165688	145485
R-25:T-30	178	137	175469	156954
R-30:T-30	206	158	196354	168405

Fig. 12 depicts task memory utilization during execution. Megabytes (MB) are the units of measurement for memory. The average memory utilization of compared methods is 147,335.67, 147,492.67, 149,286.17, 144,399.50, and 135,387.33, respectively. This indicates that the proposed technique provides more resources while using less memory. As a result, the proposed algorithm's efficiency is stated. The statistical comparison of the proposed algorithm with various algorithms in terms of cost and memory usage is shown in Table 9.

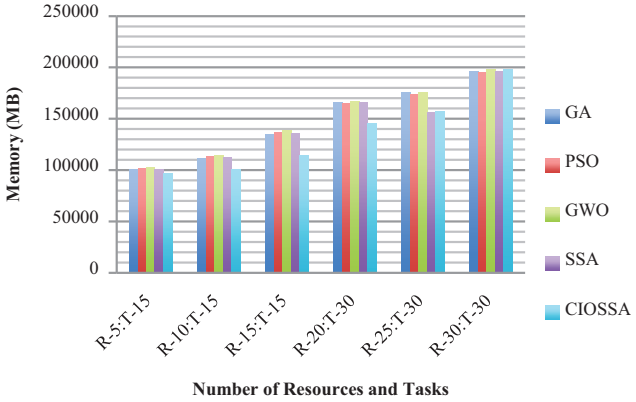


Figure 12: Comparison of memory usage

Table 9: Statistical comparison of algorithms (Cost and memory usage)

Algorithm	Mean cost ($)	Std. Deviation	Error margin	Mean memory (MB)	Std. Deviation	Error margin
GA	564	12.3	±2.3	147,335.67	1023	±184.7
PSO	624	15.4	±2.8	147,492.67	1098	±197.5
GWO	653	17.1	±3.1	149,286.17	1256	±221.4
SSA	574	10.8	±1.9	144,399.50	987	±178.3
CIOSSA	551	8.6	±1.5	135,387.33	764	±137.8

Overall, the purpose of the proposed method is to minimize cost and makespan. It's also linked to a constant μ, whose value ranges from 0 to 1. When it gets close to 0, the makespan gets more weight, and when it gets close to 1, the cost value gets more weight. When the value is somewhere in the middle of 0 and 1, both makespan and cost are equally important.

5.4 Time Complexity Analysis

Time complexity is a crucial metric for evaluating the efficiency of algorithms, particularly in cloud computing environments where large datasets and complex tasks are common. Understanding the time complexity of our all-in-one algorithm is essential for assessing its scalability and performance.

The time complexity of TSAC is $O(nlogn)$, where n is the number of tasks. This complexity arises from the hierarchical clustering process, which involves sorting and merging tasks based on their urgency. The time complexity of IOSSA is $O(m * n)$, where m is the number of iterations and n is the number of resources. This complexity reflects the iterative nature of the algorithm as it searches for optimal resource allocations. When combining TSAC and IOSSA into the CIOSSA, the overall time complexity is approximated as $O(nlogn + m * n)$. This indicates that the performance of the algorithm is primarily influenced by the clustering phase, especially as the number of tasks increases.

The time complexity of the proposed model is shown in Fig. 13. When the number of tasks is 1000, the combined model CIOSSA achieves a time complexity of 21 s, whereas TSAC and IOSSA achieve a time complexity of 40, and 80 s. When combining TSAC and IOSSA into the CIOSSA, the performance of the algorithm is primarily influenced by the clustering phase, especially as the number of tasks increases. The graph shows that CIOSSA maintains efficiency even with a high number of tasks, making it the most effective model for large-scale cloud computing environments where time efficiency and task management are critical.

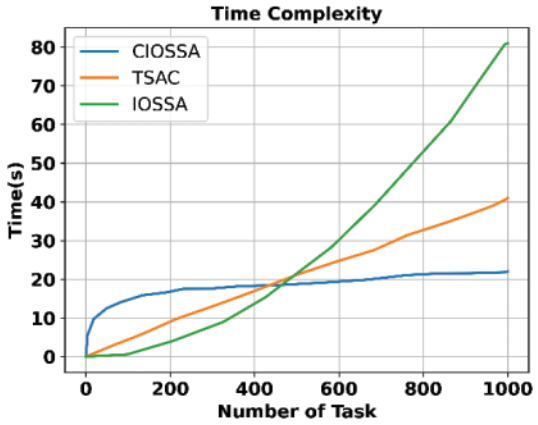


Figure 13: Time complexity of the proposed model

5.5 Technical Justification for the Improved Results Obtained by the Proposed Model

The proposed approach offers significant improvements across multiple performance metrics through its innovative task scheduling and resource allocation strategies:

Makespan: Using Task Splitting Agglomerative Clustering (TSAC) the algorithm decides which tasks should be given the higher priority based on the deadlines. Due to this prioritization, the highest-priority tasks are always scheduled as early as possible, minimizing the overall completion time of all tasks. As a result, the algorithm lowers the makespan. Then, IOSSA further optimizes the tasks through resource allocation, giving each task the best resources necessary for fast execution and further lowering the makespan.

Resource Utilization: In TSAC, the mechanism of clustering categorizes the tasks based on resource requirement needs; thus, vCPU and memory are assigned in a more appropriate way. IOSSA readjusts the resources by taking into consideration the real-time availability so that maximum utilization of available resources is assured. Due to this type of targeted allocation, the underutilization of resources is avoided, and usage becomes well-balanced and more efficient within the cloud infrastructure.

Cost: The proposed approach aims at the optimization of resources and minimization of makespan, which further directly reduces operational costs on the usage of cloud resources. Efficient use of vCPU and memory will reduce their overall consumption and lower the usage of costly resources. A reduction in makespan reduces the overall time a resource is billed for. Moreover, avoiding resource overloading by intelligent prioritizing of tasks prevents unnecessary resource additions and their subsequent expenditure.

Throughput: Resource allocation in a two-step process, followed by scheduling, allows the processing to be done at a very high speed and with great efficiency. Thus, the algorithm would allow an increase in task handling by prioritizing every task with urgency and resource requirement consideration within a given time. In turn, this throughput is accomplished due to effective processing and optimization of resource utilization.

Convergence Speed: Improved Salp Swarm Algorithm-IOT-based optimum task scheduling includes the adoption of adaptive iteration counts based on tasks and enables faster convergence towards optimal solutions. IOSSA adjusts iteration counts dynamically to make the algorithm home in on the best resource allocations quickly without extra computation. This results in quicker decision-making and faster overall scheduling processes.

Memory Usage: Hierarchical clustering has a rather neat and tidy way of segmenting tasks into priority groups, therefore diminishing the problem's complexity and reducing memory overhead in TSAC. In the same way, IOSSA optimizes the representation of task schedules and resource allocations for effective memory use. With a sorted and organized task list, the algorithm keeps away from excessive usage of memory resources; hence, it improves memory usage.

Generally, the proposed approach integrates TSAC and IOSSA, balancing makespan, resource utilization, cost efficiency, throughput, convergence speed, and memory usage. This holistic optimization framework would display superior performance in cloud scheduling environments, making it robust enough for dynamic and resource-intensive applications.

6 Conclusion and Future Work

In conclusion, enhancing scheduling efficiency and optimizing resource utilization and cost is essential for cloud computing. The methodologies, Task Splitting Agglomerative Clustering (TSAC) method improves task prioritization by categorizing tasks based on urgency, enhancing scheduling efficiency. The Input Oriented Salp Swarm Algorithm (IOSSA) uses a multi-leader approach to boost convergence speed and optimize resource allocation. Combined with the Clustered Input Oriented Salp Swarm Algorithm (CIOSSA), these methods dynamically check resource availability to minimize costs, reduce makespan, and improve overall performance in cloud computing environments. The study emphasizes the importance of efficient task scheduling in cloud computing to optimize resource allocation and reduce costs. It focuses on improving scheduling efficiency through better task prioritization and dynamic resource availability checks. The findings show a lower makespan of 18.12 s at a task of 5 and, a higher resource utilization of 74.61% at a task of 5, which indicate substantial enhancements in overall system performance, demonstrating the framework's scalability and robustness in dynamic environments. The integration of task prioritization with a multi-leader optimization approach not only enhances resource allocation efficiency but also significantly reduces makespan and operational costs in cloud computing environments.

Applicability of CIOSSA in Other Domains

The CIOSSA principles are applicable across various industries, extending beyond cloud computing to sectors such as logistics and manufacturing. The foundational elements of task organization, resource management, and optimization inherent in CIOSSA are tailored to address unique challenges in these fields.

1. **Logistics:** Optimizing task organization and resource management will be essential to making supply chains more efficient in the world of logistics. The TSAC clustering method assists in the management of logistics through the consideration of the following: urgencies and routes, which take place according to the availability of transportation resources for assessing deliveries. The logistics managers would utilize the clustering method they would group delivery tasks geographically and other requirements on the resources, which will be vehicle capacity to have the urgent shipments taken care of in good time. Dynamically allocated resources ensure there is an improvement in fleets; lower costs of operation occur because operation becomes faster, as evidenced by delivery.
2. **Manufacturing:** In manufacturing, CIOSSA assists in the smoothing of production scheduling and resource use. It categorizes the production tasks according to their unique resource needs, such as machines and personnel. The technique, therefore, concentrates on the key jobs while at the same time optimizing the general flow of work. This leads to shorter production cycles, less idle times for equipment, and higher output, thus enhancing operational efficiency.
3. **Healthcare:** CIOSSA enhances the management of available resources and patient visit scheduling in healthcare environments. Medical personnel efficiently prioritise urgent situations by evaluating patients based on their clinical needs and the severity of their diseases. This improves recovery rates and makes the best use of medical staff and facilities.

Acknowledgement: None.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: Juliet A. Murali: Write the main draft, method, concepr, figure, software; Brindha T.: Final approval of the manuscript revision and editing. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: Not applicable.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

[1] F. S. Prity, M. H. Gazi, and K. A. Uddin, "A review of task scheduling in cloud computing based on nature-inspired optimization algorithm," *Cluster Comput.*, vol. 26, no. 5, pp. 3037–3067, 2023. doi: 10.1007/s10586-023-04090-y.

[2] A. Javadpour *et al.*, "An energy-optimized embedded load balancing using DVFS computing in cloud data centers," *Comput. Commun.*, vol. 197, pp. 255–266, 2023. doi: 10.1016/j.comcom.2022.10.019.

[3] A. Javadpour, G. Wang, and S. Rezaei, "Resource management in a peer to peer cloud network for IoT," *Wirel. Pers. Commun.*, vol. 115, no. 3, pp. 2471–2488, 2020. doi: 10.1007/s11277-020-07691-7.

[4] M. O. Agbaje, O. B. Ohwo, T. G. Ayanwola, and O. Olufunmilola, "A survey of game-theoretic approach for resource management in cloud computing," *J. Comput. Netw. Commun.*, vol. 2022, no. 1, 2022, Art. no. 9323818. doi: 10.1155/2022/9323818.

[5] M. Zakarya, L. Gillam, K. Salah, O. Rana, S. Tirunagari and R. Buyya, "CoLocateMe: Aggregation-based, energy, performance and cost aware VM placement and consolidation in heterogeneous IaaS clouds," *IEEE Trans. Serv. Comput.*, vol. 16, no. 2, pp. 1023–1038, 2022. doi: 10.1109/TSC.2022.3181375.

[6] R. Aron and A. Abraham, "Resource scheduling methods for cloud computing environment: The role of meta-heuristics and artificial intelligence," *Eng. Appl. Artif. Intell.*, vol. 116, 2022, Art. no. 105345. doi: 10.1016/j.engappai.2022.105345.

[7] K. D. Thilak, K. L. Devi, C. Shanmuganathan, and K. Kalaiselvi, "Meta-heuristic algorithms to optimize two-stage task scheduling in the cloud," *SN Comput. Sci.*, vol. 5, 2024, Art. no. 122. doi: 10.1007/s42979-023-02449-x.

[8] B. Jeong, S. Baek, S. Park, J. Jeon, and Y. S. Jeong, "Stable and efficient resource management using deep neural network on cloud computing," *Neurocomputing*, vol. 521, pp. 99–112, 2023. doi: 10.1016/j.neucom.2022.11.089.

[9] A. Katal, S. Dahiya, and T. Choudhury, "Energy efficiency in cloud computing data centers: A survey on software technologies," *Cluster Comput.*, vol. 26, no. 3, pp. 1845–1875, 2023. doi: 10.1007/s10586-022-03713-0.

[10] M. N. Aktan and H. Bulut, "Metaheuristic task scheduling algorithms for cloud computing environments," *Concurr. Comput.: Pract. Exp.*, vol. 34, no. 9, 2022, Art. no. e6513. doi: 10.1002/cpe.6513.

[11] G. Natesan and A. Chokkalingam, "Task scheduling in heterogeneous cloud environment using mean grey wolf optimization algorithm," *ICT Express*, vol. 5, no. 2, pp. 110–114, 2019. doi: 10.1016/j.icte.2018.07.002.

[12] A. Pradhan, S. K. Bisoy, and A. Das, "A survey on PSO based meta-heuristic scheduling mechanism in cloud computing environment," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 8, pp. 4888–4901, 2022. doi: 10.1016/j.jksuci.2021.01.003.

[13] C. Huang and W. Yeh, "A new SSO-based Algorithm for the Bi-objective time-constrained task scheduling problem in cloud computing services," in *Distributed, Parallel, and Cluster Computing*, May 2019. doi: 10.48550/arXiv.1905.04855.

[14] J. Zhou *et al.*, "Comparative analysis of metaheuristic load balancing algorithms for efficient load balancing in cloud computing," *J. Cloud Comput.*, vol. 12, 2023, Art. no. 85. doi: 10.1186/s13677-023-00453-3.

[15] R. A. Ibrahim, A. A. Ewees, D. Oliva, M. A. Elaziz, and S. Lu, "Improved salp swarm algorithm based on particle swarm optimization for feature selection," *J. Ambient Intell. Humaniz. Comput.*, vol. 10, no. 8, pp. 3155–3169, 2019. doi: 10.1007/s12652-018-1031-9.

[16] A. A. Mohamed *et al.*, "A novel hybrid arithmetic optimization algorithm and salp swarm algorithm for data placement in cloud computing," *Soft Comput.*, vol. 27, no. 9, pp. 5769–5780, 2023. doi: 10.1007/s00500-022-07805-2.

[17] D. Rambabu and A. Govardhan, "Optimization assisted frequent pattern mining for data replication in cloud: Combining sealion and grey wolf algorithm," *Adv. Eng. Softw.*, vol. 176, 2023, Art. no. 103401.

[18] A. Kumar and S. Bawa, "Generalized ant colony optimizer: Swarm-based meta-heuristic algorithm for cloud services execution," *Computing*, vol. 101, no. 11, pp. 1609–1632, 2019. doi: 10.1007/s00607-018-0674-x.

[19] S. Singhal *et al.*, "Energy efficient resource allocation in cloud environment using metaheuristic algorithm," *IEEE Access*, vol. 11, pp. 126135–126146, 2023. doi: 10.1109/ACCESS.2023.3330434.

[20] U. K. Jena, P. K. Das, and M. R. Kabat, "Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 6, pp. 2332–2342, 2022. doi: 10.1016/j.jksuci.2020.01.012.

[21] A. Y. Hamed, M. K. Elnahary, F. S. Alsubaei, and H. H. El-Sayed, "Optimization task scheduling using cooperation search algorithm for heterogeneous cloud computing systems," *Comput. Mater. Contin.*, vol. 74, no. 1, pp. 2133–2148, 2023. doi: 10.32604/cmc.2023.032215.

[22] G. Saravanan, S. Neelakandan, S. Ezhumalai, and S. Maurya, "Improved wild horse optimization with levy flight algorithm for effective task scheduling in cloud computing," *J. Cloud Comput.*, vol. 12, 2023, Art. no. 24. doi: 10.1186/s13677-023-00401-1.

[23] S. Mangalampalli, G. R. Karri, and U. Kose, "Multi objective trust aware task scheduling algorithm in cloud computing using whale optimization," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 35, no. 2, pp. 791–809, 2023. doi: 10.1016/j.jksuci.2023.01.016.

[24] S. A. Alsaidy, A. D. Abbood, and M. A. Sahib, "Heuristic initialization of PSO task scheduling algorithm in cloud computing," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 6, pp. 2370–2382, 2022. doi: 10.1016/j.jksuci.2020.11.002.

[25] S. Tuli *et al.*, "HUNTER: AI based holistic resource management for sustainable cloud computing," *J. Syst. Softw.*, vol. 184, 2022, Art. no. 111124. doi: 10.1016/j.jss.2021.111124.

[26] D. Patel, M. K. Patra, and B. Sahoo, "GWO based task allocation for load balancing in containerized cloud," in *Proc. Int. Conf. Inventive Comput. Technol. (ICICT)*, Coimbatore, India, IEEE, 2020, pp. 655–659.

[27] S. Nabavi, L. Wen, S. S. Gill, and M. Xu, "Seagull optimization algorithm based multi-objective VM placement in edge-cloud data centers," *Internet Things Cyber-Phys. Syst.*, vol. 3, pp. 28–36, 2023. doi: 10.1016/j.iotcps.2023.01.002.