Indian Journal of Information Sources and Services
ISSN: 2231-6094 (P) Vol.14, No.2, 2024, pp.167-177

DOI: https://doi.org/10.51983/ijiss-2024.14.2.24

Optimization of System Performance through Ant Colony Optimization: A Novel Task Scheduling and Information Management Strategy for Time-Critical Applications

N.A. Suvarna[1] and Deepak Bharadwaj[2]

[1]Research Scholar, Computer Science Engineering SOES, GD Goenka University, India
[2]Associate Professor, Computer Science Engineering SOES, GD Goenka University, India
E-mail: [1]suvarna07.aradhya@gmail.com, [2]deepak.bharadwaj@gdgu.org
ORCID: [1]https://orcid.org/0000-0002-1578-4587, [2]https://orcid.org/0000-0003-0366-5449
(Received 17 April 2024; Revised 19 May 2024, Accepted 10 June 2024; Available online 28 June 2024)

Abstract **- Optimization of task scheduling and information storage/retrieval is crucial for managing resource utilization, which enhances system performance and ultimately impacts provider productivity and customer satisfaction. Efficient task scheduling aims to optimize computing time, while efficient information management focuses on maximizing memory usage. This paper presents a novel approach to task scheduling using Ant Colony Optimization (ACO) to improve time-critical objectives such as makespan and network latency, while maintaining balanced load distribution across systems. By enhancing makespan, we aim to maximize CPU utilization, and by optimizing information storage/retrieval, we target minimizing network latency. Performance across these multiple objectives is achieved by modifying the heuristic and visibility functions to guide ants toward specific solutions. The effectiveness of the proposed algorithm, Resource-Aware Load-Balancing for Time-Critical Applications (RALB-TCA), is demonstrated through implementation in the CloudSim simulation platform and benchmarking against existing techniques.**

Keywords: **Ant Colony Optimisation, Makespan, Load Balance, Network Latency, Resource Utilization, Task Scheduling**

I. INTRODUCTION

The networked computers helps in delivery of IT services through internet. Tasks are distributed for optimizing resources by matching the demands and availability. Efficient task scheduling plays a significant role as it leads to improvement in overall system performance. This is achieved through optimization of multiple objectives such as: completion time (makespan), data transfer time (network latency) and CPU utilization. It ultimately results in timeliness of services by the provider. Additionally, maintaining balance of loads across virtual machines (VMs) pave way for fault tolerant systems resulting in increased customer reliability on the provider for uninterruptable services.

Ant Colony Optimization is chosen for the study as it is a meta-heuristic, multi-objective and decentralized algorithm. It is an algorithm based on the ant's behaviour related to their search for food. It has been progressively used in task scheduling due to its ability to optimise multiple objectives and adapt to the dynamic nature of the cloud and its decentralised operation by balancing exploration and exploitation. Its decentralised decision-making helps to increase scalability and fault tolerance in cloud systems. Extensive modifications have been introduced in this algorithm by researchers in the past to seek better results. To summarize, it has the following characteristics.

- It can be applied to any general problem.
- It can explore new solutions and also utilize known solutions.
- Randomly search the solution space.
- Iteratively improve the solutions.
- It can optimize multiple parameters.
- It can work in decentralized manner.
- It is adaptive to changing environments.

In this research, we propose a metaheuristic, multi-objective, resource-aware load-balanced ant colony optimisation algorithm for time-critical applications in cloud computing. The proposed algorithm provides a pareto-optimal solution for resource allocation by minimising makespan and network latency while maintaining load balance across the entire system. The proposed algorithm is simulated using the CloudSim simulator. Experimental results are compared with previously suggested improvements for Ant Colony Optimization.

The organisation of the paper is as mentioned below. Section - II is on the related works. Section III gives insights into the proposed system and algorithm. Section IV details the simulation results obtained from the implementation and its analysis. Section V demonstrates and analyse the results, and Section VI concludes this paper.

II. Related Works

A. Background and Context

Many modern computing paradigms have emerged out of Distributed Computing, such as the Internet of Things (IoT), Mobile Computing, Edge Computing, Grid Computing, and Fog Computing (Eiriemiokhale & Olutola, 2023). They have gained popularity in recent times and have opened a number of challenges and research opportunities (De Donno et al., 2019). Task allocation and resource provisioning still remain issues in native cloud computing and its derived variants without standard norms. Numerous researchers have attempted to optimise resources using Ant Colony Optimization (ACO).

In the Ant Colony Optimization (ACO) approach, artificial ants traverse a solution space representing various scheduling possibilities. In real ants, pheromones are chemicals that are released to communicate with each other. These signals help ants coordinate their movements and work together on complex tasks, like finding the best path to a food source.

In ACO, artificial ants represent possible solutions. They deposit pheromones along their paths. Pheromones in ACO represent the intermittently updated functions that guide towards attractive solutions. The more pheromones there are, the more likely the ants will follow them. Over time, paths with higher pheromone levels become more attractive, guiding towards optimal solutions.

In ACO, pheromones let artificial ants share information and work together indirectly. This helps them explore and use solution spaces efficiently, like real ants.

ACO is very flexible and practical for solving complex problems. For example, it can manage data flow from sensors to central servers in IoT networks, optimising path selection, load balancing, and energy consumption. Combining ACO with IoT communication and security technologies makes it worthwhile in various fields (Bobir et al., 2024). Its ability to handle complex, dynamic, multi-objective, and uncertain optimisation tasks, along with its scalable and bio-inspired design, makes it valuable in logistics, transportation, telecommunications, and engineering.

Using ACO for task scheduling in large setups such as cloud environments paves the way for dynamic and adaptive solutions, considering everchanging workloads and resource availability. By imitating cooperative behaviours, the algorithm explores various solution spaces, leading to practical task assignments that enhance cloud-based operations' efficiency. Ants make decisions based on pheromone levels, indicating solution quality and heuristic information. Pheromones influence other ants' choices, and through multiple iterations inspired by the ant's behaviour, the ACO algorithm converges on an optimal or near-optimal schedule through iterative evaluation.

Multi-Objective ACO (MO-ACO) aims to optimise several objectives simultaneously using a heuristic function. The resulting Pareto Optimal Solution represents a state where resources are efficiently allocated, and any further improvements would require trade-offs in another area.

This paper implements such a multi-objective optimisation algorithm for time-critical objectives such as:

1. Makespan
2. Load Balance and
3. Network Latency

In ant colony optimisation, the visibility function is crucial in directing ants to make informed choices regarding the attractiveness of available paths. The heuristic function provides additional guidance, helping ants assess the potential quality of a solution. This knowledge complements the visibility function, enabling ants to identify which paths are worth exploring with greater precision.

The algorithm presented in this paper builds a heuristic function to improve visibility in ant colony optimisation, aiding in the selection of suitable resources (Elfarra et al., 2023). The heuristic is selected to consider the amount of available resources on individual virtual machines (VMs). Given that any cloud environment consists of heterogeneous nodes, the devised heuristic yields improved outcomes. This assertion is validated through the execution of the algorithm on cloudsim and by comparing the results with those of prior studies.

B. Literature Review

Several research papers on task scheduling through Ant Colony Optimization are studied with specific emphasis on the design of visibility and heuristic functions. The key findings are outlined below.

Tawfeek et al., (2013) proposed a heuristic function, which is the inverse of the execution speed of the task on a particular VM. So, the lesser the execution speed, the higher the probability of choosing that VM. The objective function is set to minimise only makespan.

The visibility function is given by:

$$\eta_{ij} = 1/d_{ij} \quad (1)$$

Where,

$$d_{ij} = TaskLength/ExecSpeed + FileSize/BandWidth \quad (2)$$

The pheromone update rule is set as follows:

$$\Delta\tau_{ij}^{k}(t) = Q/L^{k}(t). \quad (3)$$

Where k is the iteration number, $\Delta\tau_{ij}$ is the incremental pheromone deposited by the ant while assigning task i to resource j, Q is a pheromone constant, and L is the expected makespan for the tour. The comparison is made with basic algorithms such as Round Robin and FCFS to demonstrate the improved performance.

Guo, (2017) proposes to improve both makespan and communication overhead by defining the objective function in terms of linear weights and heuristic function to choose the most suitable

VM is:

$$\eta_{ij} = Load_j * 1/et_{ij} \quad (4)$$

Where et_{ij} is the execution time of task j on i^{th} machine. And $Load_j$ is given by

$$Load_j = 1 - (E_j - E_{avg}) / \sum_{j \in VM} E_j \quad (5)$$

Where E_j is the execution time of the virtual machine, and E_{avg} is the average execution time of the Virtual machines.

Chandrashekar et al., (2023) presented a hybrid weighted ACO. The author defines the heuristic function as,

$$\eta_{ij} = \alpha * \text{makespan} + \beta * \text{cost} \quad (6)$$

The makespan of the individual VM is calculated as follows:

$$Makespan(M_{ij}) = S_{ij} + WT_{ij}(t) + EC_{ij}(t) \quad (7)$$

Where S is the time of submission of the task to the VM, WT is the waiting time, and EC is the execution time of the task. Then, the final makespan is calculated as per the following equation:

$$\text{Makespan} = \sum_{i=1}^{n} \sum_{j=1}^{m} (M_{ij}) \quad (8)$$

Liu et al., (2019) employ the genetic algorithm for initialising the pheromone and a search algorithm based on ACO, which facilitates accelerated convergence. Subsequently, to mitigate the risk of the algorithm being trapped in local optima, they suggest the random rule for selecting the subsequent node along the path. Furthermore, they propose a weighted constraint function that considers both time and cost. To determine the weighting of cost and time, they utilise the Analytic Hierarchy Process (AHP).

Lin et al., (2019) use ACO-MCMS (Multi-Objective of Container Microservice Scheduling) to consider the optimisation of transmission overhead, load balancing, and reliability as a measure of the failure rate of the servers. The authors suggested a scheduling algorithm to solve container microservices in clusters.

Chaharsooghi et al., (2008) utilised a Multiple Objective Evolutionary Algorithm to enhance scheduling operations by minimising costs and time while optimising resource utilisation and balancing the load. They employed the Tencent model for cost calculation for bandwidth and flow rate. However, the algorithm demonstrated poor resource utilisation.

Moon et al., (2017) implemented slave Ant Colony Optimization (SACO), which is a diversification of probability parameters for slave ants, reducing their dependency on other ants. With updated makespan information, one ant is designated as a typical ant, exhibiting the best makespan for the group. The remaining ants become slave ants. Subsequently, a local pheromone is updated.

Selvan et al., (2009) modelled the scheduling problem as a directed acyclic graph (DAG), with tasks representing nodes and edges indicating task transmissions between nodes. This graph representation imposes task dependencies and precedence rules, stating the sequence of subtask completion. The authors propose Ant Colony Optimization (ACO) to minimise both makespan and latency overheads. The heuristic function wraps both computation and latency costs.

Sharma & Garg, (2022) introduced a QoS-based task scheduling using ACO by modifying the load balancing factor in the heuristic function in comparison to (3) as:

$$LB_j = 1 - (E_j - E_{avg}) / (E_j + E_{avg}) \quad (9)$$

Junyu & Lichen, (2018) recommended a simple heuristic function:

$$RT = \sum_{i=1}^{k} RT_i \quad (10)$$

Where RT is the Runtime and K corresponds to the number of jobs on the specified virtual machine. Then, the final runtime is selected as the maximum runtime, and the other VMs are made to wait for the slowest/overloaded processor to finish (Santhosh & Prasad, 2023).

Zuo et al., (2015) proposed performance and budget optimisation through ACO (PBACO) to improve makespan, resource utilisation, and user costs. The authors provide a technique for the problem of ACO's solution falling into the local optima through a fitness function based on performance and cost. This fitness function is then used to evaluate the quality of possible solutions. The fitness function is:

$$Fit(x) = \gamma * e^{-F(x)} + \delta * e^{-B(x)} \quad (11)$$

Where γ and δ are the performance and cost weight factors, and F(x) and B(x) are the performance and cost functions, respectively.

In summary, the referenced studies considered the execution speed or the load when choosing the VM. Considering both simultaneously led to better results in terms of makespan, load balancing and resource utilisation. Considering only execution speed in the visibility function in Ant Colony Optimization (ACO) could lead to biased decision-making by the artificial ants. This bias will result in suboptimal load balancing because other important factors, such as resource availability and data required for tasks, are not taken into account. Again, solely focusing on load in the visibility function may overlook other factors that affect task execution, such as execution speed or resource availability. Leading to an unbalanced utilisation of resources.

While (4), (5), and (9) work upon execution speed, their heuristic functions fall short of encompassing the broader search space. Previous researchers in (Tawfeek et al., 2013; Chandrashekar et al., 2023; Moon et al., 2017; Junyu &

Lichen, 2018) primarily focused on minimising makespan, resulting in only partial load balancing. In contrast, (Liu et al., 2019; Chaharsooghi & Kermani, 2008) employed evolutionary algorithms to expand the search space. Additionally, (Lin et al., 2019) utilised the Analytic Hierarchy Process (AHP) as a statistical method to predict node failures and enhance reliability. Furthermore, the study by (Selvan et al., 2009) is dedicated to reducing the costs. García et al., (2024) is theoretical work on ACO for parallel computation, and (Scianna, 2024; Okrah et al., 2024; Xu et al., 2023) are implementations of ACO for specific applications.

C. Comparative Analysis

The types of modifications possible to ACO, along with its objectives and effects on task scheduling, are summed up in TABLE I below.

TABLE I POTENTIAL MODIFICATIONS IN ACO

	Modification	**Objective**	**Impact**
1	Pheromone update rule	Improves convergence speed and solution quality	Introduces dynamic pheromone updates based on task characteristics
2	Local search heuristics	Enhances the exploitation of local search space	Incorporates additional heuristics for local optimisation
3	Task priority	Addresses task priority in scheduling	Introduces priority-based decision rules for task allocation
4	Dynamic parameter adaption	Improves algorithm adaptability and performance	Adapts dynamically based on the evolving workload
5	Hybridisation with heuristic methods	Combine ACO with other optimisation techniques	Integrate ACO with heuristics for better performance
6	Multi-objective optimisation	Optimise multiple objectives simultaneously	Extends ACO for multiple parameters
7	Communication overhead reduction	Minimise communication overhead	Address networking challenges

In accordance with TABLE I, the current state of the specified modifications is discerned in each of the cited papers and succinctly presented in TABLE II for expeditious comparison of the literature review.

III. PROPOSED METHOD

A. Problem Statement

This research paper proposes an innovative approach to address the multi-objective task scheduling problem using Ant Colony Optimization through modified visibility and heuristic functions. The proposed methodology involves adopting a new strategy for visibility and heuristic functions and setting up pheromone update rules to allocate tasks based on resource availability(threshold) across virtual machines (VMs). The primary objective is to enhance three critical time-related parameters: makespan, network latency, and load balance.

TABLE II COMPARISON OF LITERATURE WORKS

Ref	**Algorithm**	**Modifications**						
		Pheromone Update Rule	Local Search	Task Priority	Dynamic Parameter Adaption	Hybrid Heuristics	Multi-objective	Communication overheads
[2]	ACO	X	X	X	✓	X	X	X
[3]	MO-ACO	X	X	X	X	X	✓	✓
[4]	HWACOA	X	✓	X	✓	X	✓	✓
[5]	TCLB-GAAC	X	✓	X	✓	✓	✓	X
[6]	ACO-MCMS	X	✓	X	✓	X	✓	✓
[7]	MORAP	X	✓	X	✓	X	✓	X
[8]	SACO	X	✓	X	✓	X	✓	X
[9]	PARALLEL - ACO	X	✓	✓	✓	X	✓	✓
[10]	LBACO	X	✓	X	✓	✓	✓	✓
[11]	IMPROVED ACO	X	X	X	✓	X	X	X
[15]	PBACO	X	✓	X	✓	✓	✓	✓

B. Mathematical Formulation

The proposed Enhanced Load-Balanced ACO (ELB-ACO) framework is based on the following assumptions regarding the cloud infrastructure:

- Tasks are assumed to be queued up at the broker level and are assigned to available virtual machines (VMs) for execution in batches. The size of the task queue is varied from 10 to 1000.
- While the actual cloud environment comprises a vast number of VMs, only a subset of it is considered in this study.
- The heterogeneous nature of the cloud is depicted through variations in the number of processors, execution speeds, and bandwidth capacities.
- Tasks varying in length and file sizes are considered within the framework.

In every batch of tasks queued for processing at the broker, the following variables are defined.

Number of tasks: n

Number of virtual machines (VMs): m

Execution speed of the i^{th} VM: ES_i in MIPS

Number of processors in the i^{th} VM: P_i

Length of the jth task: L_j in MIs

Load share of the i^{th} VM: LS_i in MIs

Computing load of the i^{th} VM: $Load_i$ in MIs

Where,

$$Load_i = \sum_{k \in i} L_k \quad (12)$$

The computing power of i^{th} VM = $ES_i * P_i = CP_i$ MIPS

$$LS_i = \sum_{j=1}^{n} L_j * (CP_i / \sum_{i=1}^{m} CP_i) \quad (13)$$

Similarly,

File size of j^{th} task: FS_j Bits

Bandwidth of i^{th} VM (machine): BW_i in kbps

Transmission Time of j^{th} task to i^{th} VM: T_{ij}

Where,

$$T_{ij} = FS_j / BW_i \text{ seconds}$$

Transmission load of i^{th} VM: TR_i bits

Where,

$$TR_i = \sum_{k \in i} T_{ik} \quad (14)$$

The transmission share of i^{th} VM (machine) is,

$$TS_i = \sum_{j=1}^{n} FS_j * (BW_i / \sum_{i=1}^{m} BW_i) \quad (15)$$

The heuristic function is set as:

$$\eta_{ij} = 0.5 * (LS_i - L_i)/LS_i + 0.5 * (TR_i - T_i)/TR_i \quad (16)$$

Giving equal weightage to both execution time and transmission time.

The initial pheromone is set as:

$$\tau_{ij} = (ES_i * P_i) \quad (17)$$

The basic pheromone update rule is set as follows:

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \sum_{k \in ants} \Delta\tau_{ij}^{k} \quad (18)$$

Where ρ is the pheromone evaporation rate, and $\Delta\tau_{ij}$ is the amount of pheromone deposited by ants on edge (i, j).

The transition rule for choosing the next available resource for allocation is:

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{k \in allowed} \tau_{ik}^{\alpha} \cdot \eta_{ik}^{\beta}} \quad (19)$$

Where $P_{ij}(t)$ is the probability of moving from node i to node j at time t, α and β are parameters controlling the influence of pheromones and heuristic information, and η_{ij} is the heuristic information. Proposed Algorithm shown in fig. 1 below.

C. Proposed Algorithm

Algorithm 1: Proposed Algorithm - Resource-Aware Load Balancing for Time-Critical Applications (RALB-TCA)

Initialize the Parameters
for each VM_i (i ranging from 1 to m)
{Computing Power of VM_i = CP_i = $ES_i * P_i$
Load Share of VM_i = LS_i
= $\sum_{j=1}^{n} L_j * (CP_i / \sum_{i=1}^{m} CP_i)$
For each Task T_j (j ranging from 1 to n)
Transmission Time for T_j = FS_j / BW_i
Initialial Pheromone,
$\tau_{ij}(0) = 1/(ES_i * P_i)$
Transmission Share of VM_i = TS_i
= $\sum_{j=1}^{n} FS_j * (BW_i / \sum_{i=1}^{m} BW_i)$
}
Start
{for each iteration (ITER)
{ for each ant (A)
{ for each task T_j (j ranging from 1 to n)
Select VM with highest Pheromone
Insert the seelected task in tabu list
$allowed_k$ ={0,1,…,n-1}-$tabu_k$
Update pheromone (eq. (16))
Update heuristic (eq. (14))
}
Compute makespan
Compute total transmission time
Clear the tabu list
}
Choose the best results
}
End

Fig. 1 Proposed Algorithm

The basic ACO is modified with respect to the visibility function η_{ij} and the initial pheromone τ. The artificial ants, unlike real ants, are not blindfolded. They are made intelligent in choosing the resources through heuristic function. The heuristic in this paper is modified to accommodate two main time-critical parameters for optimisation and load balancing. The factors considered are:

1. Completion Time (Makespan).
2. Data Transfer Time (DTT).

A linear function with an equal weightage of 50% for each of these two parameters is formed as in (16).

The formula adapted ensures that the computing load on any VM does not exceed its capacity (processing share). At the same time, it also ensures that the data to be transferred does not exceed the transmission capability of the VM. Also, the load allocated (computing load or transmission load) is considered in the calculation. The heuristic value is also checked at every step to ensure that it does not become negative. If the value goes negative, it is changed to zero value.

The flowchart of the proposed algorithm is explained in Figure 2 below.

D. Proposed Model

The cloud environment is simulated using a small cluster of the cloud infrastructure to implement the algorithms under investigation. The physical cloud system encompasses a vast collection of geographically dispersed computing devices networked together. In our study, we focus on a subsection or cluster of this network located at a single data centre, comprising a limited number of computing nodes. Cloudlets or tasks are dynamically batch-processed in real-time, with centralised task allocation occurring at the broker level. The framework of the representative cloud infrastructure for implementation is indicated in Figure 3.

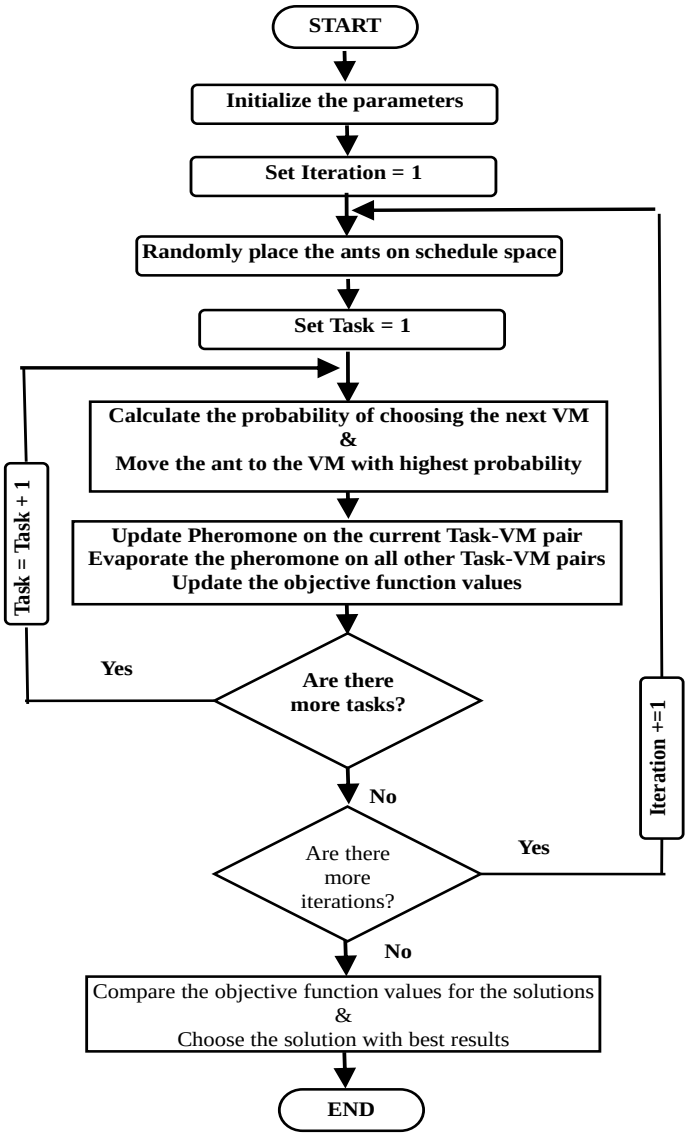

Fig. 2 Flowchart of the proposed algorithm

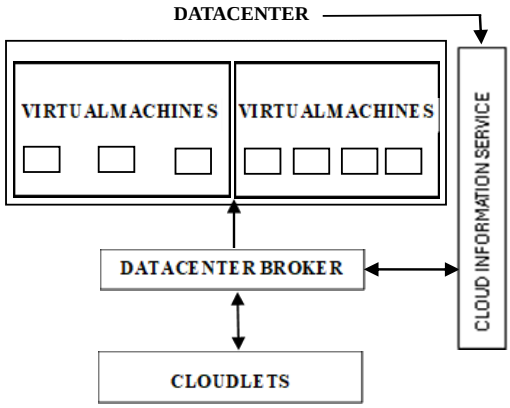

Fig. 3 Framework of Cloud Environment for Implementation

IV. Experimental Results

The algorithm offered in this study is instantiated and executed utilising the Cloud Sim Toolkit, a Java-based simulation framework designed for the modelling of cloud computing environments. The simulated infrastructure encompasses data centres, brokers, and virtual machines, each endowed with virtualised computing resources that holistically represent the pertinent entities in the simulated cloud ecosystem.

The tasks enqueued within the broker are systematically allocated for execution across the extant virtual machines in accordance with the prescribed algorithm. In order to substantiate the superior efficacy of the proposed algorithm, an identical set of tasks is subjected to the ACO algorithm delineated in [3], and the ensuing outcomes derived from these two algorithmic frameworks are explicated through graphical representation in the ensuing figures.

The task quantity varies across three distinct data sizes: 10, 100, and 1000. The task lengths are randomly assigned within the range of 1,00,000 to 50,00,000 million instructions.

The three Virtual Machines are set with the important parameter values as indicated in TABLE III below.

TABLE III COMPUTING POWER OF VMs

VM	Execution Speed (MIPS)	Processors	Computing Speed (MIPS)
1	1024	1	1024
2	2048	2	4096
3	3072	3	9216

TABLE IV indicates the resulting task distribution among the three available virtual machines (VMs) when subjected to two algorithms: the algorithm outlined in (Guo, 2017). and the novel approach proposed in this study.

TABLE IV DISTRIBUTION OF TASKS AMONG VMs

Tasks	VM	Number of Tasks	
		RALB-TCA	ACO
10	0	1	7
	1	1	1
	2	8	2
100	0	27	52
	1	36	22
	2	37	26
1000	0	191	927
	1	381	14
	2	428	59

A. Makespan

The time required by each virtual machine (VM) to complete the execution of its allocated tasks is detailed in TABLE V below, categorised by data size. The makespan metric is obtained from this table. The makespan, implying the maximum duration experienced by the slowest virtual machine (VM), introduces a consequential waiting interval for other VMs which have completed executing their tasks in advance. This metric serves as a key indicator, offering awareness about job completion times.

TABLE V COMPLETION TIME OF TASKS

Tasks	VM	Completion Time in Seconds	
		RALB-TCA	ACO
10	0	308.2	16666.0
	1	242.5	879.0
	2	2353.4	252.8
100	0	8497.8	15360.0
	1	2814.4	1895.7
	2	1226.4	872.0
1000	0	399820.0	1949095.0
	1	200132.9	8297.9
	2	100247.5	13365.9

By considering the maximum completion time for each data size from TABLE V, the makespan metric is derived as shown in TABLE VI below.

TABLE VI MAKESPAN

Tasks	Makespan in Seconds		Makespan in Minutes	
	RALB-TCA	ACO	RALB-TCA	ACO
10	2353.4	16666.0	39	278
100	8497.8	15360.0	142	256
1000	399820.0	1949095.0	6664	32485

The data provided in TABLE VI, illustrating the makespan in minutes, is graphically depicted in Figure 4 for data sizes 10, 100, and 1000. The graphs demonstrate the reduction in makespan achieved by the proposed algorithm across all data sizes when compared to the referenced algorithm (Guo, 2017). The improved makespan is an indication of improved CPU throughput.

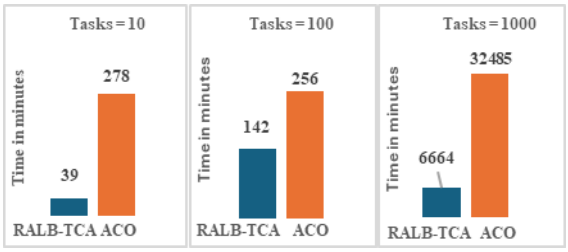

Fig. 4 Makespan

B. Data Transfer Time (Network Latency)

Tasks are defined by an attribute known as "File Size," which signifies the volume of data to be transmitted during task execution. Similarly, each virtual machine (VM) is defined by a parameter known as "Bandwidth," denoting the maximum achievable data transfer rate between the VM and other components within the cloud infrastructure, including storage devices, other VMs, or external networks. By correlating the file size of a task with the bandwidth of a VM, the duration required for data transfer to complete the task is established. This data transfer time is also called "Communication Overhead" or "Network Latency".

The data transfer time consumed by each virtual machine (VM) as a consequence of task allocation per the algorithm is presented in TABLE VII below. The results are listed for each data size comprising tasks in numbers 10, 100, and 1000.

TABLE VII DATA TRANSFER TIME OF TASKS

Tasks	VM	Data Transfer Time in Seconds	
		RALB-TCA	ACO
10	0	16.0	223.0
	1	0.5	128.0
	2	112.0	32.0
100	0	3186.0	3739.0
	1	1245.5	704.8
	2	229.1	408.0
1000	0	20059.0	95062016.0
	1	9496.0	1753088.0
	2	4917.4	2062306.0

Data transfer time denotes the temporal investment essential for transmitting task execution essential files from the broker to the virtual machine (VM). In heterogeneous environments, VMs exhibit disparate bandwidth capacities.

In a heterogeneous environment characterised by virtual machines with varying bandwidth capacities and tasked with a diverse set of operations entailing varied data transfer durations, the VM that concludes data transfer most expeditiously may encounter a waiting period until all other VMs complete their respective tasks. This scenario inherently contributes to network latency, determined by the VM experiencing the most extended data transfer duration. Consequently, TABLE VIII is derived from TABLE VII to delineate network latency.

TABLE VIII DATA TRANSFER TIME (NETWORK LATENCY)

Tasks	DTT in Seconds		DTT in Minutes	
	RALB-TCA	ACO	RALB-TCA	ACO
10	112	223	2	4
100	3186	3739	53	62
1000	20059	95062016	334	1584366

The data provided in TABLE VIII, illustrating the network latency in minutes, is graphically depicted in Figure 5 for data sizes of 10, 100, and 1000. The graphs clearly demonstrate the improved network latency achieved by the proposed algorithm across all data sizes compared to the referenced algorithm (Guo, 2017).

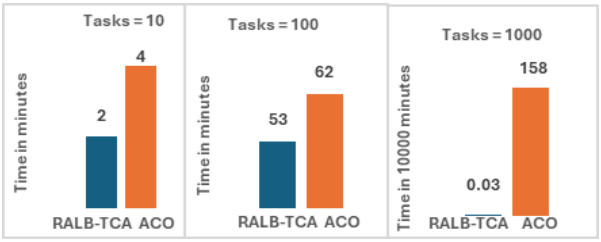

Fig. 5 Network Latency

C. CPU Utilisation

CPU utilisation factor in cloud computing refers to the measure of the extent to which the central processing unit (CPU) of a virtual machine or physical server is utilised over a given period of time. With our proposed methodology, we aim to propose optimal utilisation of the CPU.

CPU utilisation factor provides insights into the efficiency and performance of the system. High CPU utilisation indicates that the CPU operates close to its maximum capacity, which may lead to performance degradation or resource contention. On the other hand, low CPU utilisation suggests that the CPU is underutilised, which may indicate inefficient resource allocation or provisioning. Our algorithm, along with prioritising time-critical solutions, also demonstrates optimal CPU utilisation across VMs.

TABLE IX is the result obtained for CPU utilisation against the available computing time. Mathematical calculations involved to arrive at the available computing time are mentioned below for a sample case.

TABLE IX UTILISATION OF CPU

Tasks	VM	Computing Time in Seconds		
		Available	Utilised	
			RALB-TCA	ACO
10	1	1642677	315575	17066376
	2	6570710	993469	3601002
	3	14784097	21688441	2330107
100	1	2252276	8701733	15728753
	2	9009103	11527637	7764563
	3	20270483	11302492	8038546
1000	1	153788680	409415764	1.996E+09
	2	615154722	819744706	33988255
	3	1384098124	923881056	123180106

The computation of available computing time for each Virtual Machine (VM) is determined through the following steps:

1. Compute the total length of tasks in Millions of Instructions (MI).
2. Determine the execution speed of each VM in Millions of Instructions Per Second (MIPS).
3. Calculate the time required for task completion by dividing the value obtained in step 1 by the value obtained in step 2.
4. Determine the Load Share in MIs that each VM can execute for the duration of time obtained in step 3.

1) Sample Calculation for 100 Tasks

Total task lengths amount to 31,531,862 Million Instructions (MIs). The computing speed of each Virtual Machine (VM) is calculated by multiplying the execution speed by the number of processors.

VM1 has a computing speed of 1,024 MIPS with one processor.

VM2 has a computing speed of 4,096 MIPS with two processors.

VM3 has a computing speed of 9,216 MIPS with three processors.

The total computing speed equals 14,336 MIPS.

The time required for completion is determined by dividing the total task lengths by the total computing speed, resulting in 2,199 seconds.

The load share of VM1 is calculated by multiplying its computing speed (1,024 MIPS) by the time required for completion (2,199 seconds), yielding 2,252,275.9 MIs.

Now, TABLE X is an extension of the calculations of Table IX to express the utilisation of computing time as a percentage of the available power to arrive at a degree of utilisation.

TABLE X DEGREE OF UTILISATION OF COMPUTING POWER

Tasks	VM	RALB-TCA	ACO
10	1	20%	1040%
	2	20%	50%
	3	150%	20%
100	1	390%	700%
	2	130%	90%
	3	60%	40%
1000	1	270%	1300%
	2	130%	10%
	3	70%	10%

The graphical representation of the data of TABLE X, depicting the degree of computing time used by the CPUs of virtual machines, is illustrated in the following Figures 6, 7, and 8.

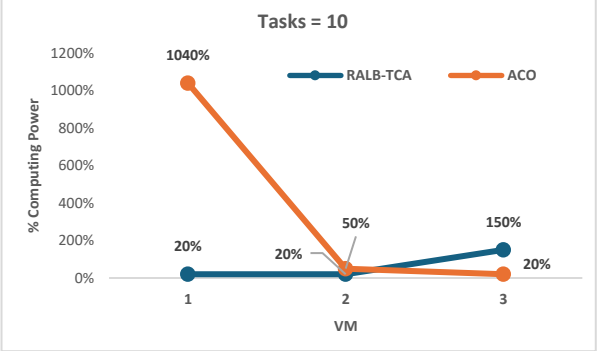

Fig. 6 Degree of Utilisation of Computing Power for Ten Tasks

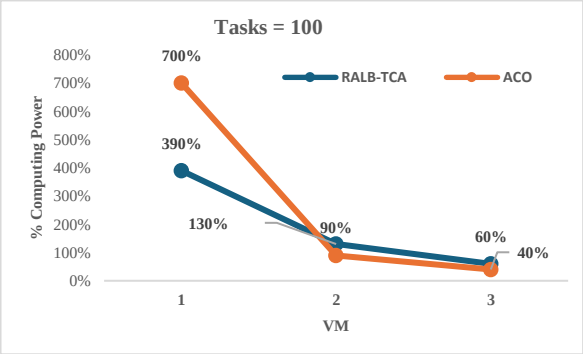

Fig. 7 Degree of Utilisation of Computing Power for 100 Tasks

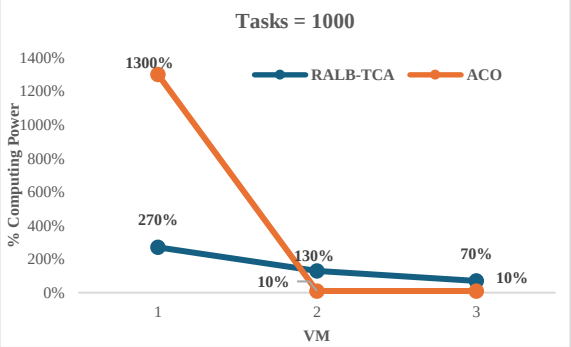

Fig. 8 Degree of Utilisation of Computing Power for 1000 Tasks

In the pursuit of harmonising computational and transmission loads while concomitantly endeavouring to minimise the makespan, virtual machines (VMs) do not achieve full exploitation of their operational capacities. The undesirable consequence of deviating from optimal utilisation manifests as an underutilisation of other available VMs. The graphical representation in Figure 6, Figure 7, and Figure 8 illustrates that, across diverse data sizes, the curve associated with the proposed algorithm exhibits a more plateaued profile when contrasted with the conventional Ant Colony Optimization (ACO). This characteristic denotes a more equitable workload distribution among VMs, thereby enhancing the system's overall efficiency.

V. DISCUSSION

The experimental findings demonstrated enhancements in the performance of the RALB-TCA algorithm concerning both completion time (CPU throughput) and data transfer time (network latency) associated with the transfer of requisite files for task execution while maintaining load balance across virtual machines. The time-critical metrics such as makespan, CPU utilisation (throughput), and network latency showed drastic improvements compared to the improvements referenced to ACO (Guo, 2017). In a cloud setup with virtual machines having drastic variations in characteristics, the balance in the loads is essential to lower the amount of waiting time for the slowest machine to complete its operation. The plateaued graphs indicate improved balance in computing time and network latency.

The experiment assigned equal weightage to these parameters. However, tailoring them to the specific demands of an application is likely to yield superior outcomes. The additional computational overhead incurred in pre-execution heuristic function calculations may be disregarded when dealing with large datasets, as is commonly encountered in cloud computing.

Moreover, prioritising tasks based on their length, particularly by addressing longer tasks first, can lead to a more balanced distribution of workloads and improved resource use. This method has the potential to reduce idle times and enhance throughput.

Incorporating elements like job prioritisation, deadlines, fault tolerance, and service level agreements (SLAs) can improve the performance of meta-heuristic algorithms. It can be integrated with RALB-TCA in real-world applications. These factors allow for more flexible and adaptive scheduling, enhancing the algorithm's ability to handle complex and large-scale cloud computing scenarios.

VI. CONCLUSION

The exposition of the research presented in this document points out the efficiency of the RALB-TCA algorithm, which improves different facets of task scheduling for cloud computing. The new threshold-based algorithm that introduces a visibility and heuristic function to be modified by it is discussed in detail. Through our work, we paid particular attention to some performance metrics—makespan, data transfer time, load balance and CPU utilisation.

The modified implementations and algorithmic improvements have proven adept at efficiently allocating tasks and minimising communication overhead, significantly reducing data transfer times. This is particularly advantageous for applications with large datasets and intricate task dependencies. Changes in deployment, along with algorithmic improvements, have proven to enhance task allocation effectiveness without reducing the time taken to transfer data. These modified implementations would, therefore, be more appropriate to use for applications that deal with large datasets as well as task dependencies due to their success.

To sum up, we find evidence suggesting that the RALB-TCA algorithm can drive improved system performance. Prospective efforts might delve into more tweaks and practical adoptions of this proposed algorithm within different cloud computing landscapes, seeking optimal resource utilisation while meeting user quality requirements under varied operational conditions.

REFERENCES

[1] Bobir, A.O., Askariy, M., Otabek, Y.Y., Nodir, R.K., Rakhima, A., Zukhra, Z.Y., & Sherzod, A.A. (2024). Utilizing Deep Learning and the Internet of Things to Monitor the Health of Aquatic Ecosystems to Conserve Biodiversity. *Natural and Engineering Sciences, 9*(1), 72-83.

[2] Chaharsooghi, S. K., & Kermani, A. H. M. (2008). An effective ant colony optimization algorithm (ACO) for multi-objective resource allocation problem (MORAP). *Applied mathematics and computation, 200*(1), 167-177.

[3] Chandrashekar, C., Krishnadoss, P., Kedalu Poornachary, V., Ananthakrishnan, B., & Rangasamy, K. (2023). HWACOA scheduler: Hybrid weighted ant colony optimization algorithm for task scheduling in cloud computing. *Applied Sciences*, *13*(6), 3433. https://doi.org/10.3390/app13063433.

[4] De Donno, M., Tange, K., & Dragoni, N. (2019). Foundations and evolution of modern computing paradigms: Cloud, iot, edge, and fog. *IEEE access*, *7*, 150936-150948.

[5] Eiriemiokhale, K. A., & Olutola, J. B. (2023). Application of the Internet of Things for quality service delivery in Nigerian university libraries. *Indian Journal of Information Sources and Services*, *13*(1), 17-25.

[6] Elfarra, B.K., Salha, M.A., Rasheed, R.S., Aldahdooh, J., & Abusamra, A.A. (2023). Enhancing the Lifetime of WSN Using a Modified Ant Colony Optimization Algorithm. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, 14*(3), 143-155.

[7] García, M., López, N., & Rodríguez, I. (2024). A full process algebraic representation of Ant Colony Optimization. *Information Sciences*, *658*, 120025. https://doi.org/10.1016/j.ins.2023.120025.

[8] Guo, Q. (2017). Task scheduling based on ant colony optimization in cloud environment. *In AIP conference proceedings*, *1834*(1).

[9] Junyu, Y., & Lichen, Z. (2018). The Load Balanced Ant Colony Optimization based on Cloud Computing. *International Conference on Network, Communication, Computer Engineering (NCCE)*, ATLANTIS PRESS.

[10] Lin, M., Xi, J., Bai, W., & Wu, J. (2019). Ant colony algorithm for multi-objective optimization of container-based microservice scheduling in cloud. *IEEE access*, *7*, 83088-83100.

[11] Liu, L., Luo, T., & Du, Y. (2019). A new task scheduling strategy based on improved ant colony algorithm in IaaS layer. In *2019 International Conference on Computer, Information and Telecommunication Systems (CITS)*, 1-5.

[12] Moon, Y., Yu, H., Gil, J. M., & Lim, J. (2017). A slave ants based ant colony optimization algorithm for task scheduling in cloud computing environments. *Human-centric Computing and Information Sciences*, *7*, 1-10.

[13] Okrah, S. K., Wiah, E. N., Otoo, H., & Addor, J. A. (2024). A velocity-based ACO algorithm for optimizing routes and social cost. *Scientific African*, *23*, e02031. https://doi.org/10.1016/j.sciaf.2023.e02031

[14] Santhosh, G., & Prasad, K. V. (2023). Energy Saving Scheme for Compressed Data Sensing Towards Improving Network Lifetime for Cluster based WSN. *Journal of Internet Services and Information Security, 13*(1), 64-77.

[15] Scianna, M. (2024). The AddACO: A bio-inspired modified version of the ant colony optimization algorithm to solve travel salesman problems. *Mathematics and Computers in Simulation*, *218*, 357-382.

[16] Selvan, T. V., Chitra, P., & Venkatesh, P. (2009). Parallel implementation of task scheduling using ant colony optimization. *International Journal of Recent Trends in Engineering*, *1*(1), 339-343.

[17] Sharma, N., & Garg, P. (2022). Ant colony based optimization model for QoS-Based task scheduling in cloud computing environment. *Measurement: Sensors*, *24*, 100531. https://doi.org/10.1016/j.measen.2022.100531.

[18] Tawfeek, M. A., El-Sisi, A., Keshk, A. E., & Torkey, F. A. (2013). Cloud task scheduling based on ant colony optimization. *In 8th International Conference on Computer Engineering & Systems (ICCES)*, 64-69.

[19] Xu, G., Lin, H., Cheng, Y., & Li, S. (2023). An Improved Ant Colony Optimization for Solving Task Scheduling Problem in Radar Signal Processing System. *Journal of Signal Processing Systems*, *95*(2), 333-350.

[20] Zuo, L., Shu, L., Dong, S., Zhu, C., & Hara, T. (2015). A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing. *IEEE Access*, *3*, 2687-2699.