Iraqi Journal for Computer Science and Mathematics

Journal Homepage: http://journal.esj.edu.iq/index.php/IJCM
e-ISSN: 2788-7421 p-ISSN: 2958-0544

Adaptive Heterogeneity Index Cloudlet Scheduler for Variable Workload and Virtual Machine Configuration

D Gritto[1]*, P Muthulakshmi[2]

[1]Department of Computer Science, College of Science and Humanities, SRM Institute of Science and Technology, Chengalpet, Tamilnadu, India-603203.
[2]Department of Computer Science, College of Science and Humanities, SRM Institute of Science and Technology, Chengalpet, Tamilnadu, India-603203,

*Corresponding Author: D Gritto

DOI: https://doi.org/10.52866/ijcsm.2024.05.03.048
Received April 2024; Accepted June 2024; Available online August 2024

ABSTRACT: Cloud environments often exhibit varying levels of heterogeneity arising from the diverse characteristics of cloudlets and virtual machines. This research paper focuses on addressing this heterogeneity and proposes two scheduling algorithms: the Variance Managed Heuristic Scheduler (VMHS) and the Adaptive Heterogeneity Index Cloudlet Scheduler (AHICS). AHICS aims to minimize makespan, virtual machine underutilization, the degree of load imbalance, and the deviation of completion time among virtual machines. AHICS serves as the main scheduler, while VMHS and MaxMin function as sub-schedulers in this proposed work. This multi-objective AHICS scheduling algorithm harnesses the strengths of both schedulers. AHICS adaptively selects either VMHS or MaxMin based on the heterogeneity level of the cloudlets and virtual machines, employing VMHS in low heterogeneity scenarios and MaxMin in high heterogeneity scenarios. Implemented using the CloudSim 3.0.3 simulator, the AHICS scheduler outperforms other heuristic scheduling algorithms, including MinMin, TASA, HAMM, PTFR, and RSSM. Experimental results demonstrate improvements of 3-5% in makespan, 4-6% in virtual machine utilization, 25–84% in load imbalance, and a reduction of 25–91% in completion time deviation for both low and high heterogeneity scenarios. These performance gains can translate into substantial cost savings, increased efficiency, and an improved user experience.

Keywords: Cloudlets-Virtual Machines-Scheduling-Load Balancing-Heterogeneity Index.

1. INTRODUCTION

The resource requirements of both businesses and individual users in recent times have exhibited dynamic trends. Cloud computing has emerged as a convenient solution to meet this growing demand. Cloud computing provides access to computing resources and services via the internet. Instead of owning and managing physical infrastructure, users can access virtual instances of resources, including storage, servers, databases, software, and networking, from various CSPs [1]. The resources housed in the data centers can be rapidly provisioned and de-provisioned to meet these dynamic requirements. By leveraging cloud services like Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), cloud users can efficiently host, deploy, and operate a diverse range of applications in a dynamic manner.

In any service-based computing environment, performance pertains to the effectiveness of a system or application in managing user tasks and adhering to SLA terms. The SLA outlines performance measures, specifically the QoS, to be delivered by the CSP to the cloud users [2]. One of the crucial factors behind the attainment of the SLA and QoS is the resource management technique. Cloud resource management covers activities like allocating, monitoring, and optimizing resources such as VMs, storage, and networking utilities to meet various application needs. In this context, cloud resource management tasks such as virtualization, scheduling, load balancing, VM migration, and VM consolidation become essential. These techniques work together to improve resource usage, the effective delivery of cloud services, and performance. However, managing cloud resources and achieving optimal cloud service delivery pose widespread challenges. These challenges stem from various factors, such as the dynamic nature of user demand and constraints dictated by measures listed in the SLA, specifically the attainment of QoS. Other key challenges include resource contention, network latency, scalability, virtualization overheads, continuous monitoring and optimization, etc.

Scheduling and load balancing are fundamental techniques for optimizing performance by achieving QoS metrics like makespan reduction, resource utilization, energy efficiency, adaptability, and cost reduction in cloud environments. Scheduling involves making decisions about when (time) and where (host or VM) to execute the cloudlet or task. In general, scheduling involves two types: virtual machine scheduling and cloudlet scheduling. Virtual machine scheduling maps VMs to hosts, while cloudlet scheduling maps cloudlets to VMs. The cloudlet scheduling algorithm performs resource matching by analyzing the resource requirements of the cloudlets and the configurations of the VMs.

The resource requirements of a cloudlet typically include the number of processing elements (CPU cores), memory (RAM), storage, bandwidth, and other relevant factors. Similarly, the configuration of VMs encompasses attributes such as CPU capacity, available memory, disk space, network bandwidth, and many other relevant parameters. By comparing the resource requirements of cloudlets with the capacities of VMs, the scheduling algorithm identifies the most suitable VM for executing each cloudlet.

Load balancing focuses on distributing user or network workloads evenly across the available hosts or VMs during the allocation or scheduling phase to avoid bottlenecks, overutilization, and underutilization of resources. Both are complementary strategies that work together to ensure that cloudlets are allocated to VMs in a way that optimizes performance and solves many of the aforementioned issues [3]. Distributed environments manage load distribution using static and dynamic load balancing algorithms. Static algorithms rely on predetermined rules based on statistical data about resources like hosts and VMs. These rules consider characteristics like processing speed, number of cores, and memory before workload allocation. They are well-suited for predictable workloads with consistent traffic patterns. Dynamic load balancing algorithms, on the other hand, adapt to real-time workloads. They handle fluctuating traffic patterns by continuously assessing the current state of the system, including CPU usage, memory availability, and other metrics. This allows them to adjust task assignments dynamically to maintain optimal resource utilization [4].

Apart from scheduling and load balancing, other techniques involved in efficient resource management are virtualization, virtual machine migration, and consolidation. Virtualization is the core technology that enables the on-demand and dynamic management of cloud resources. The two main approaches to achieving virtualization are hardware and software virtualization. The foundation of hardware virtualization is the hypervisor, which virtualizes the entire hardware layer of a machine and creates virtual machines. Each VM runs its own operating system, providing a high degree of isolation and flexibility. Software virtualization, on the other hand, virtualizes the operating system layer and creates containers. Containers share the host machine's operating system kernel but isolate the application and its dependencies. Containerization tools like Docker manage container creation. Other types of resource virtualization include server, storage, network, desktop, application, and data virtualization. VM migration involves moving a running VM from one physical machine to another without interrupting its operation. VM consolidation groups VMs on fewer physical machines to free up underutilized machines. VM consolidation is often used in conjunction with VM migration. Both are done during load balancing and server maintenance or to improve resource utilization, cost reduction, energy consumption, disaster recovery, downtime minimization, and other critical benefits. Figure 1 illustrates cloudlet or user task scheduling and execution.

The clients are the end-users or applications of the system, who submit the cloudlets to the datacenter broker for execution. The datacenter broker acts as an intermediary between the clients and the datacenters. It manages operations like receiving cloudlets from clients, placing them in a queue, allocating resources in the datacenter, and scheduling them to be executed on the VMs. The datacenter broker matches client requirements or demands with suitable datacenters, negotiates SLAs, and manages the provisioning of datacenter resources. Hosts are physical machines that run VMs. VMs are self-contained software environments that run on a host and provide a platform for running cloudlets. Cloudlets are small pieces of code or user tasks submitted by clients. The cloudlet queue is where cloudlets are stored before they are assigned to a VM. The datacenter is the physical location where the resources reside. The CIS holds a repository of information about available resources, services, and configurations within the cloud environment. The load balancer distributes incoming traffic among the different VMs in the datacenter. The datacenter broker selects the VM and cloudlet scheduling policies (VMScheduler SpaceShared, VMScheduler TimeShared, CloudletScheduler SpaceShared, CloudletScheduler TimeShared, etc.). Scheduling policies are the guidelines or principles that govern the overall scheduling approach. Scheduling algorithms are a set of well-defined steps that determine the order in which cloudlets are to be mapped and processed. The datacenter broker maps the VMs to the hosts and cloudlets to the VMs based on the selected scheduling policies and algorithms. The VM scheduler policies define the rules for allocating VMs within the physical hosts, optimizing resource usage and energy efficiency, etc. VM scheduler algorithms implement these policies, analyzing VM and host characteristics to determine VM placement. Cloudlet scheduler policies govern cloudlet assignment to the VMs based on the cloudlet requirements and objectives, like makespan reduction, increased resource utilization, load balancing, etc. Cloudlet scheduler algorithms implement these policies to execute the cloudlets in the VMs.

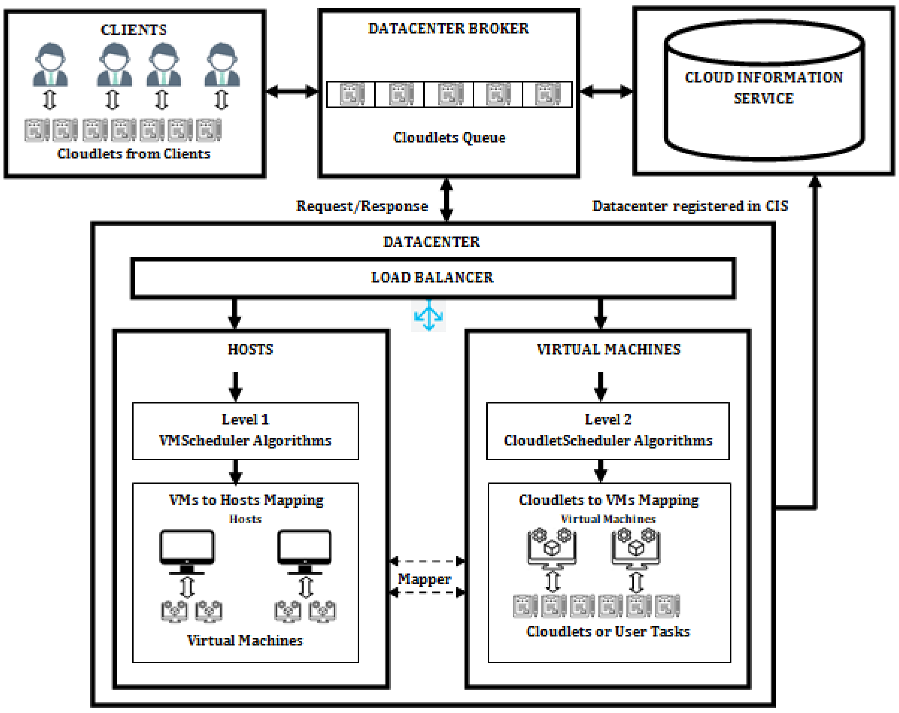


Figure 1: **Typical Cloudlet Scheduling Model**

Cloudlet scheduling is essential for optimizing cloud performance, but it faces challenges due to the heterogeneity among cloudlets and VMs. Heterogeneity refers to the varying characteristics of cloudlets and VMs in terms of processing speed (MIPS), cloudlet size (MI), memory, bandwidth, etc. Most scheduling algorithms fail to effectively address this heterogeneity, which can negatively impact cloud system performance and efficiency. The motivation for this research lies in the need to optimize cloud performance, often hindered by the neglect of heterogeneity in existing scheduling algorithms. This paper proposes the Adaptive Heterogeneity Index Cloudlet Scheduler (AHICS) algorithm as a solution. AHICS is based on the Variance Managed Heuristic Scheduler (VMHS) and MaxMin sub-schedulers. By considering the combined heterogeneity index (CHetInx), AHICS adaptively selects either VMHS or MaxMin, employing VMHS in low heterogeneity scenarios and MaxMin in high heterogeneity scenarios. The AHICS algorithm aims to improve resource allocation, enhance makespan, VM utilization, load balance, reduce the deviation of completion time among the VMs, and address other heterogeneity-related challenges. AHICS significantly outperforms existing heuristic algorithms like MinMin, TASA, HAMM, PTFR, and RSSM. The contributions of this AHICS scheduler are as follows:

- **Makespan reduction:** The AHICS scheduler effectively reduces the overall completion time or makespan of the schedule by considering the heterogeneity, leading to improved cloudlet processing time.
- **Improved VM utilization:** By carefully matching cloudlets and VMs based on their characteristics, AHICS enhances VM utilization, reducing resource wastage and improving the cost-efficiency.
- **Minimized load imbalance:** AHICS evenly distributes the cloudlets among VMs, preventing any single VM from being overloaded while others remains idle or underloaded. This ensures efficient resource utilization and prevents system bottlenecks.
- **Reduced completion time deviation:** The scheduler strives to minimize the variation in completion time among the VMs, leading to more predictable and consistent performance, which is often desirable in cloud environments.

This paper is organized into seven standard sections. Section 1 introduces the foundational theory for scheduling and load balancing in cloud environments, the underlying motivations, and the contribution of this paper. Section 2 examines existing similar research works. Section 3, the framework and formulation section, provides the system model, problem statement, problem formulation, objectives, and the computational formulas behind the proposed work. Section 4 presents the proposed algorithms, namely VMHS, MaxMin, and AHICS schedulers, along with their time complexity analyses. Section 5 explains the experimental section. Section 6 analyzes the results and provides a discussion on the outcomes of the experimentation. Finally, Section 7 presents the paper's conclusion.

2. RELATED RESEARCH WORK

The most common challenges in cloud computing include makespan reduction, QoS, resource, energy optimization, and fault tolerance. Several research studies documented in the literature propose scheduling algorithms to address these issues. These formulations are generally classified as heuristic scheduling, meta-heuristic scheduling, and hybrid scheduling algorithms capable of optimizing either a single or multiple objective(s).

Furthermore, cloudlets can be broadly categorized into two main types: dependent cloudlets and independent cloudlets. Independent cloudlets are self-sufficient and do not require any specific order or interaction with other cloudlets to execute. They can be processed in any order without impacting the outcome. This makes them ideal for parallel processing tasks that can be broken down into independent units. Dependent cloudlets, on the other hand, rely on the successful completion of other cloudlets before they start execution. They have a defined order of execution and might require data or results from previously executed cloudlets. They are similar to workflows where the cloudlets need to be scheduled in a specific sequence to ensure correct execution.

Section 2.1 reviews heuristic, meta-heuristic, and hybrid cloudlet scheduling algorithms. Section 2.2 then reviews related research work about load balancing. This review focuses on papers concerning reducing makespan, improving resource utilization, and achieving load balancing. Table 1 enumerates the various abbreviations and expansions used in this paper.

2.1 Review of Cloudlet Scheduling Algorithms

Effective cloudlet-to-VM mapping and load balancing is crucial for optimizing performance in cloud environments. This review explores the existing techniques, their strengths, weaknesses, and contributions to the field. HEPGA proposed by *H. Mikram et al.*, is a hybrid algorithm that combines HEFT, PSO, and GA algorithms. This workflow-based HEPGA algorithm aims to minimize makespan, cost, and latency while maximizing resource utilization. The HEFT algorithm provides cloudlet priority, PSO utilizes Levy's distribution for exploration to find solutions, and the genetic algorithm generates refined solutions using selection, mutation, and crossover operations. The algorithm performs better than GA, HGA, and PSO algorithms. The performance may deteriorate as workflows become larger and more complex. Incorrect parameter settings could result in suboptimal solutions [6]. The adaptive cloudlet PSO-ACO cloudlet scheduling algorithm, proposed by *N. R. Sabat et al.*, aims to reduce makespan and cost. The algorithm also strives to find better fitness values by achieving reduced completion time and waiting time. PSO is used to generate the initial solution, and ACO refines the solution by optimizing cloudlet distribution to VMs through pheromone distribution. This approach outperforms both PSO and ACO algorithms individually. The effectiveness of the algorithm depends on carefully tuning parameters in both PSO and ACO [7]. *B. Rambabu et al.* implemented the least-trailed MBO and a hybrid MBO-ABC algorithm to optimize resource utilization, makespan, and degree of load imbalance. While MBO showed promise, it was found to have limited exploration capabilities. Therefore, a hybrid approach combining MBO with the ABC algorithm was implemented. The hybrid MBO-ABC algorithm demonstrated superior performance compared to the standalone MBO, achieving better optimization results and exhibiting improved convergence, effectively addressing the limitations identified in MBO. Comparing the MBO-ABC algorithm solely with MBO could be insufficient for a comprehensive performance assessment [8].

The GEC-DRP algorithm, proposed by *K. L. Devi et al.*, comprises several modules with specific functionalities: task clustering, workload prediction, VM controller, resource provisioner, and task scheduler. Task clustering utilizes K-means clustering to categorize the tasks based on their CPU, memory, and bandwidth demands, resulting in K clusters. Workload prediction is achieved through exponential smoothing. The VM controller determines the number of VMs required based on the arrival rate of each cluster. The resource provisioner allocates resources for the VMs within the host. The GEC-DRP algorithm improves metrics such as makespan, load imbalance, throughput, and cost. However, it is important to note that these performance enhancements come at the cost of increased time complexity. This limits its scalability and applicability to large-scale computing environments [9]. The SAAC approach, proposed by *A. A. Nasr et al.*, is a two-phase optimization strategy that combines SA and ACO algorithms to improve scheduling efficiency. In the first phase, SA generates a random solution, which serves as the starting point for the subsequent ACO scheduling phase. ACO then attempts to find a better or optimal solution compared to the initial one generated by SA. By combining SA and ACO, SAAC demonstrates superior performance compared to using either SA or ACO alone. Comparing the SAAC algorithm only with SA or ACO might not provide a comprehensive assessment of its performance [10]. *K. Kamalam et al.* combines the MinMin and MaxMin scheduling algorithms. The cloudlets are queued in increasing order of their length. If the average completion time exceeds the standard deviation of completion times, MinMin is used. MaxMin is used in the alternative case. The PTFR suffers from a potentially high makespan, especially with the late arrival of large cloudlets [11]. RSSM, proposed by *N. M. Reda et al.*, integrates the range suffrage and sort mid scheduling algorithms to minimize makespan and enhance resource utilization. Each iteration involves a two-phase cloudlet selection process. First, cloudlets satisfying the range selection criteria are grouped into a subset based on their suffrage value. Then, the sort mid approach is applied to this subset in the second phase. The completion times of the selected cloudlets are sorted in non-decreasing order. For each cloudlet, two consecutive mid-completion times, K and K+1 (where K=m/2), are identified, and their average is calculated. The cloudlet with the highest average value is then assigned to the VM with the shortest completion time. While the algorithm achieves optimal performance, it comes at the cost of high time complexity. This can limit the applicability of RSSM in large-scale computing environments [12, 13, and 14].

Table 1: List of Abbreviations and Expansions

Abbreviation	*Expansion*	*Abbreviation*	*Expansion*
ABC	Artificial Bee Colony	HEPGA	HEFT-PSO-GA
ACO	Ant Colony Optimization	HLFO	Hybrid Lyrebird Falcon Optimization
ACOFTF	ACO File Type Format	IABC-TS	Improved Artificial Bee Colony-Task Scheduling
ACOLBA	Ant Colony Optimization based Load Balancing Algorithm	IWOA	Improved Whales Optimization Algorithm
AHICS	Adaptive Heterogeneity Index Cloudlet Scheduler	LFO	Lyrebird Falcon Optimization
AWS	Amazon Web Services	LOA	Lyrebird Optimization Algorithm
CA-MLBS	Content Aware Machine Learning Based Load Balancing Scheduler	MBO	Monarch Butterfly Optimization
CNN	Convolutional Neural Network	ML	Machine Learning
CSLBA	Crow Search Algorithm for Load Balancing	MPSO	Modified Particle Swarm Optimization
CSP	Cloud Service Provider	MTBLB	Modified Multi Time Based Load Balancing
D2B	Dynamic Degree Balance	OLB	Opportunistic Load Balancing
DDMTS	DRL-based Dynamic Multi-Objective Task Scheduling	PSO	Particle Swarm Optimization
DFTF	Data Files Type Formatting	PTFR	Paired Task Front Rear
DOA	Dingo Optimization Algorithm	QMPSO	Q-Learning Modified Particle Swarm Optimization
DQL	Deep Q-Learning	QoS	Quality of Service
DQN	Deep Q-Networks	RL	Reinforcement Learning
DRL	Deep Reinforcement Learning	RR	Round Robin
FOA	Falcon Optimization Algorithm	RSSM	Range Suffrage Sort Mid
FOA	Falcon Optimization Algorithm	SA	Simulated Annealing
GA	Genetic Algorithm	SAAC	Simulated Annealing-Ant Colony
GEC-DRP	Genetic Encoded Chromosomes-Dynamic Resource Provisioner	SLA	Service Level Agreement
GWO	Grey Wolf Optimization	SVM	Support Vector Machine
HAMM	Hybrid Algorithm of MinMin and MaxMin	TASA	Task Aware Scheduling Algorithm
HDDB	Hybrid Dynamic Degree Balance	VM	Virtual Machine
HEFT	Heterogeneous Earliest Finishing Time	VMHS	Variance Managed Heuristic Scheduler

The HAMM cloudlet scheduling algorithm proposed by *I. Syed et al.*, combines MinMin and MaxMin strategies to optimize metrics like makespan, waiting time, resource utilization, and load balancing. It dynamically categorizes cloudlets into small and large categories based on the average cloudlet length. The algorithm then dynamically allocates the cloudlets: MinMin scheduling is used when there are fewer small cloudlets, while MaxMin scheduling is employed when there are more small cloudlets. However, its reliance on MinMin can sometimes lead to increased makespan and reduced resource utilization [15]. The extended MinMin algorithm of *J. Y. Maipan-Uku et al.* processes cloudlets in their arrival order. In each iteration, it calculates the maximum and minimum completion times from the completion time matrix. The algorithm then compares the completion time of the first cloudlet in the queue with the maximum completion time. If the completion time of the first cloudlet is shorter, the MinMin scheduling approach is used to allocate the cloudlet to the VM with the minimum completion time. Otherwise, MaxMin scheduling is employed. The performance impacted by the late arrival of large-size cloudlets, resulting in increased makespan [16].

The deadline-based minimum completion time algorithm, devised by *M. Kumar et al.*, efficiently distributes workloads across VMs by elastically provisioning or de-provisioning resources. This decision-making process considers factors like the average number of missed deadlines and recent historical data, i.e., the last k optimal interval features. The VM threshold value based on load is a critical parameter in determining the elastic measures. This approach aims to minimize missed deadlines and makespan. However, its performance degrades when the interval is above 15 [17]. The HCA algorithm proposed by *J. P. B. Mapetu et al.*, aims to allocate the maximum number of cloudlets within the minimum number of VMs. Load balancing is achieved by attaining the closest completion time among all VMs. The cloudlet allocation occurs in two phases. The allocation in the first phase is done using the heuristic of optimal completion time. In the second phase, the allocation is based on the earliest finishing time of the VMs. However, the HCA algorithm ultimately exhibits behavior identical to the MaxMin scheduling algorithm [18].

2.2 Review of Load Balancing Algorithms

A. R. Khan et al. proposes a method for dynamic load balancing and optimization of multiple QoS metrics using cloudlet scheduling. This work leverages ML techniques and optimization principles to achieve these goals. The CNN estimates VM load levels by analyzing metrics like CPU and memory utilization. The RNN utilizes these estimates to analyze the trends and patterns in VM load over time to predict future loads. RL employs a load threshold to cluster the VMs as overloaded or underloaded. The clustering efficiency is further improved by integrating RL with HLFO, a hybrid of LOA and FOA algorithms. HLFO, using multi-objective optimization, aims to improve factors like makespan, energy consumption, load balancing, CPU and memory usage, and cloudlet prioritization. It achieves better results compared to RL, LOA, or FOA algorithms. The model is intricate and requires significant computational resources and time. Insufficient details on the algorithms and hyper-parameters hinder the ability to replicate the results [19]. The DDMTS algorithm, proposed by *Z. Tong et al.*, employs DRL and DQN in order to achieve load-balanced cloudlet scheduling. The DRL model selects a suitable VM for the cloudlet based on the VM load limits. The cloud platform model decides whether the allocation will violate the SLA or not. If the SLA is not violated, a positive reward is assigned. Otherwise, the cloudlet is rejected. The deadline is the primary SLA constraint taken into consideration. This DQN enables the DRL agent to effectively learn the optimal dynamic load balancing policy in the DDMTS algorithm. The comparative study shows that the algorithm minimizes load imbalance and rejection rate compared to OLB, RR, and random algorithms. The DDMTS algorithm is limited by the complexity of training the DRL model and may require significant computational resources [20].

QMPSO, implement by *U. K. Jena et al.*, is a hybrid of MPSO and improved Q-learning algorithms. The improved Q-learning stores only the Q-value of the best action for each state in order to reduce storage space. It determines the load of each VM and tries to minimize the load imbalance among VMs. It is integrated with MPSO to enhance load balancing and the convergence rate. The algorithm has resulted in reduced makespan, waiting time, energy consumption, load imbalance, and improved throughput compared to MPSO and Q-learning strategies. The QMPSO algorithm may be limited by the accuracy of its Q-value estimation [21]. The CA-MLBS algorithm, proposed by *M. Adil et al.*, balances the workload among the VMs by identifying the content type of the workloads. It utilizes a SVM classifier to categorize the cloudlets as text, images, audio, or videos. The algorithm operates in two phases: the first phase involves content-based cloudlet classification, while the second phase focuses on load balancing. Based on the content type, the algorithm creates four groups of VMs, and cloudlets are scheduled for the VMs in each category accordingly. The cloudlet scheduling is performed using a PSO scheduler. The proposed algorithm outperforms existing approaches such as DFTF and ACOFTF in terms of makespan, throughput, and response time. The CA-MLBS algorithm may be limited by the accuracy of its SVM classifier, especially for complex or hybrid content types [22].

The multi-level hybrid load balancing algorithm of *Elsakaan et al.* aims to achieve load balancing by utilizing two-level schedulers: global and local schedulers. The K-means clustering algorithm is implemented to cluster the datacenter servers into four clusters based on server utilization ratio and makespan. At the next level, the RR algorithm distributes the workload among the clusters. The GA is used to distribute cloudlets among the servers within a cluster. The load balancing module identifies overloaded and underloaded servers or datacenter clusters to migrate the workload to the most underutilized ones. This methodology aims to reduce makespan, response time, SLA violations, and the number of cloudlet migrations at runtime. The computational overhead, sensitivity to cluster formation, potential for suboptimal solutions, and limited adaptability are the limitations of this work. [23]. *R. Vijay et al.* combined GA and PSO to optimize resource allocation in cloud computing. It aims to minimize completion time, cost, waiting time, makespan, and improve load balancing. The GA leverages evolutionary theory to generate a promising mapping of cloudlets to VMs after a predetermined number of iterations. This initial mapping is then further refined by PSO using a fitness function. By utilizing the strengths of both algorithms, this approach seeks to optimize the aforementioned metrics. The algorithm may result in local minima and limited consideration of Quality of Service (QoS), which are also limitations [24]. The IABC-TS algorithm, proposed by *R. N. Hanuman et al.*, is used to achieve load balancing by considering queue length, cloudlet length, and VM capacities as parameters of the fitness function. The algorithm dynamically monitors the workload on each VM and manages an even load distribution among the VMs. IABC-TS is designed to be suitable for both homogeneous and heterogeneous cloud setups. This multi-objective optimization tries to minimize makespan, cost, and improve load balancing. The IABC-TS algorithm may struggle with complex workload patterns or sudden changes in resource demand [25].

The GWO-PSO algorithm, proposed by of *M. S. Al Reshan et al.*, aims to achieve fast convergence and global optimization in load balancing by combining the strengths of the GWO and PSO algorithms. The exploration, exploitation, and convergence capabilities of GWO are combined with the global optimization capability of the PSO algorithm to attain the objective. It tries to avoid getting trapped in local minima. In comparison to PSO, SSO, ABC, BAT, and GWO, the GWO-PSO produces an optimal makespan, response time, throughput, and load balancing. The GWO-PSO algorithm may be sensitive to parameter tuning and may struggle with highly dynamic workloads [26]. Table 2 presents a comparison of scheduling and load balancing algorithms.

Table 2: **Comparison of Cloudlet Scheduling and Load Balancing Algorithms**

Category	***Algorithm***	***Description and Prime Metrics Optimized***	***Advantages***	***Disadvantages***
Evolutionary Algorithms [27]	Bacteria Foraging Task Scheduling Algorithm (BFTSA)	BFTSA maps cloudlets to VMs to minimize energy consumption and makespan while ensuring load balancing. PMO: Energy consumption, makespan and load balancing.	By preventing from local minima, the algorithm minimizes makespan and energy consumption while improving load balancing.	The priority of the cloudlet is not set with any special mechanism.
Hybrid Algorithm [28]	SLA Load Balancing Algorithm for the Datacenters (SLA-LB)	It predicts potential SLA violations in datacenters and migrates the VMs. PMO: Makespan, execution time, resource utilization and SLA violation.	It offers dynamic scheduling and load balancing and ensures SLA compliance. It improves makespan and execution time.	Work load migration procedure is not discussed.
Hybrid Algorithm [29]	Clustering based MaxMin Scheduling	Groups the cloudlets by resource needs and execution time variation. It prioritizes scheduling cloudlets to the clusters with the longest completion time. PMO: Makespan, VM utilization.	It improves the overall makespan compared to MaxMin by efficiently utilizing faster VMs for longer tasks and handling variable task lengths through clustering.	Clustering VMs and simulating cluster adds overhead. The effectiveness is based on the clustering method selected.
Hybrid Algorithm [30]	Task Aware Scheduling Algorithm (TASA)	Leveraging the strengths of suffrage and MinMin algorithms. For even numbered tasks, suffrage is used, while for odd numbered tasks, MinMin is used. PMO: Makespan, response time.	Reduced makespan, response time and efficiency in resource allocation.	Potential overhead due to continuous switching of scheduling algorithms. Performance may vary based on the number of cloudlets in real environments.
Hybrid Algorithm [31]	Fuzzy Round Robin Algorithm (FRR)	It considers processing speed and the current VM load. It tracks VM details and assigns incoming requests to the VMs with the least load. PMO: Average response time, Datacenter processing time.	Outperforms RR by reducing processing time and improving overall response time, maximizing resource utilization.	Complexity of the algorithm. The assessment of cloudlet characteristics is not discussed.
Hybrid Evolutionary Algorithms [32]	Hybrid Dingo and Whale Optimization Algorithm Load Balancing Mechanism (HDWOA-LBM)	Combining DOA and IWOA, HDWOA achieves near-optimal load balancing. PMO: Load balancing, reliability, resource utilization, makespan and throughput.	The dynamic adaptability of this algorithm produces near-optimal and efficient load balancing.	Complexity of implementation, fine-tuning of parameters is not discussed in detail, though it is a crucial aspect of optimization algorithms for improving performance.
Hybrid Evolutionary Algorithms [33]	Osmotic Hybrid Artificial Bee Colony and Ant Colony Optimization Algorithm (OH-BAC)	Combining ABC, ACO algorithms, and osmosis principles it selects the hosts for the VM migration. PMO: SLA violation, performance degradation, number of VM migrations, and number of host shutdowns.	Achieves higher energy efficiency compared to other bio-inspired algorithms taken for comparison. Minimizes SLA violations, performance degradation, VM migrations, and host shutdowns.	Has higher SLA violations time per active host. Combining the knowledge bases of ABC and ACO adds complexity.

PMO[*]: Prime Metrics Optimized.

The CSLBA algorithm, proposed by *H. Singh et al.*, is inspired by the intelligent foraging behavior of crows. It aims to optimally assign cloudlets to VMs. It leverages concepts like probability and flight length from the crow search algorithm to iteratively explore solutions and update the best solution. The performance of the algorithm is evaluated against the standard ACOLBA algorithm. Results demonstrate that CSLBA outperforms ACOLBA in reducing average power consumption, cost, and balancing data center loading under different service broker policies, including closest data center and optimization for response time. The CSLBA algorithm converges within a fair number of iterations and results in enhanced results for various VM configurations. The CSLBA may be limited by its reliance on the accuracy of the crow search algorithm [34]. MCS-DQN, proposed by *A. Chraibi et al.*, aims to minimize makespan using RL and DQN. It employs a reward function to enhance convergence and performance. The algorithm demonstrates optimal results in minimizing makespan, cloudlet waiting time, and improving resource utilization. However, it requires more computational resources than traditional heuristic algorithms. The algorithm's efficiency is dependent on its training parameters [35]. The HDDB algorithm, proposed by *A. Joshi et al.*, is a hybrid algorithm based on the D2B-CPU and D2B-membership algorithms. The D2B-CPU prioritizes placing VMs on hosts with lower CPU utilization in order to balance CPU load and achieve even load distribution. D2B-membership estimates a membership value for each host and allocates VMs to hosts with suitable membership values. The membership value considers factors like requested RAM, MIPS, and bandwidth by the VM and the available RAM, MIPS, and bandwidth of the host. The performance of HDDB is reported to be better than FCFS, SJF, and RR algorithms in terms of turnaround time, throughput, load balancing, cost, CPU, bandwidth, and memory utilization. The HDDB algorithm may struggle with dynamic workloads and may not be as effective in handling highly heterogeneous cloud environments [36].

P. Kumar et al. proposed the MTBLB algorithm, which utilizes three different approaches in order to achieve load balancing. AWS CloudWatch is instrumental in collecting requests from user VMs. Cron Jobs handles the processing of these requests in a systematic manner. The integration of these components with the RR algorithm ensures efficient load balancing and optimal resource utilization in cloud environments. The algorithm performs better than the RR algorithm in terms of the standard deviation of response time. The dependencies on AWS CloudWatch and Cron Jobs services could introduce potential points of failure [37].

While the reviewed research articles had not explicitly addressed heterogeneity as a primary focus, the proposed AHICS algorithm offers several key advantages. Its direct consideration of heterogeneity, computational efficiency, and simplicity of implementation make it a promising candidate for optimizing cloud performance.

3. FRAMEWORK AND FORMULATION

3.1 System Model

This paper tackles the challenge of heterogeneity in cloud environments. This heterogeneity introduces challenges in resource allocation and scheduling decisions, which impact factors like makespan, VM utilization, and load balancing. To address this, a three-tier scheduling approach is proposed.

The architecture consists of an AHICS scheduler, a heterogeneity index estimator, a VMHS scheduler, and a MaxMin scheduler. AHICS acts as the main coordinator, monitoring the level of heterogeneity through the heterogeneity index estimator within the cloud. The heterogeneity index estimator component calculates the degree of variation within the cloud environment. When heterogeneity is low (CLHetInx), the VMHS scheduler efficiently allocates cloudlets to VMs. The MaxMin scheduler can handle highly heterogeneous (CHHetInx) environments, where there are greater variations in cloudlet and VM characteristics.

The AHICS scheduler switches between the VMHS and MaxMin schedulers based on the combined heterogeneity index (CHetInx) between the cloudlets and the VMs. This dynamic selection aims to optimize scheduling metrics such as makespan, VM utilization, degree of load balancing, and reduce the standard deviation of completion time among VMs. Figure 2 illustrates the system model of the proposed work.

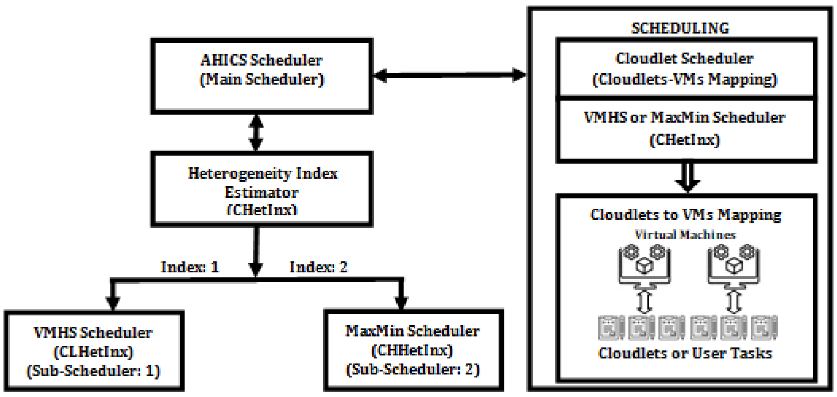


Figure 2: **System Model of AHICS Scheduler**

3.2 Problem Definition

Existing cloud scheduling algorithms are not effective in addressing the heterogeneity among cloudlets and VMs, leading to suboptimal makespan, resource utilization, and performance. This paper develops a novel AHICS scheduling algorithm that can effectively handle heterogeneity and improve the efficiency of cloud computing systems. The scheduler is designed to adapt to varying levels of heterogeneity within the cloud.

3.3 Problem Formulation

The cloud environment comprises a set of n cloudlets CDL={cd_1, cd_2, cd_3… cd_n} and m virtual machines VML={vm_1, vm_2, vm_3 … vm_m} where n, m $\in$ Z^+. The cloudlets may also have varying IDs, lengths, execution times, priorities, and deadlines. All cloudlets in this study are independent, non-preemptive and have zero arrival time i.e., all arrive at the same time. The VMs have distinct configurations, including processing power (MIPS), number of processors, memory, bandwidth, storage, virtual machine manager (VMM), etc. The proposed VMHS, MaxMin and AHICS schedulers aim to map the cloudlets to VMs by satisfying the objectives outlined in Section 3.4. This allocation is based on the heterogeneity index among the cloudlets and the VMs. The computation for the heterogeneity index is elaborated in Section 3.5.

3.4 Objectives

The primary objectives of the AHICS scheduler are as follows:

1. Minimize Makespan: Minimize the overall completion time of cloudlets.
2. Maximize VM Utilization: Efficiently allocate cloudlets to VMs to fully utilize available VMs.
3. Ensure load balancing: Distribute cloudlets evenly across VMs by preventing overloading or underloading of any specific VM.
4. Minimize the standard deviation of cloudlet completion time among the VMs.

3.5 Computational Formulas

The formulas involved in computing various scheduling metrics and the heterogeneity index calculation are described below. This heterogeneity index calculation is used within the heterogeneity index estimator of the proposed model. The AHICS scheduler leverages this estimate to select either the VMHS or MaxMin scheduler. Table 3 contains the various notations used in these formulas and their descriptions.

Table 3: List of Notations and Descriptions

Notation	*Description*	*Notation*	*Description*
AVM_{ur}	Average VM utilization ratio	MCT_c	Mean Completion Time of cloudlet
CDL	Cloudlet List	MI	Million Instructions
CHetInx	Combined Heterogeneity Index	MIPS	Million Instructions per Second
CHHetInx	Combined High Heterogeneity Index	MS_{SL}	Makespan or Schedule Length
CLHetInx	Combined Low Heterogeneity Index	MVMct	Mean VM completion time
Covcd	Co-variance of cloudlets	MVMS	Mean VM Speed
Covvm	Co-variance of VMs	n	Number of Cloudlets
$CT_c[n][m]$	Completion Time of cloudlet	$RT_{vm}[m]$	Ready Time of VM
$CT_{vm}[j]$	Completion Time of VM	SDCDct[i]	Standard Deviation of Cloudlet completion time
DL_{IB}	Degree of Load Imbalance	SD_{CDL}	Standard Deviation of Cloudlet Length
$ET_c[n][m]$	Execution Time of cloudlet	SDVMct	Standard Deviation of VM completion time
$HetInx_{cd}$	Heterogeneity Index among cloudlets	SD_{VMS}	Standard Deviation of VM Speed
$HetInx_{vm}$	Heterogeneity Index among VMs	TCT_c	Total Completion Time of cloudlet
m	Number of VMs	VML	VM List
MCL	Mean Cloudlet Length	MCT_c	Mean Completion Time of cloudlet

Execution time of a cloudlet (ET_c[][]) depicts the actual execution or computation time of the ith cloudlet on the jth VM. It is defined by the ratio of cloudlet length to the processing speed of the VM.

$$ET_c[i][j] = \frac{\text{Length of Cloudlet}_i\,(\text{in MI})}{\text{Processing Speed of VM}_j\ (\text{in MIPS})}\ \text{measured in msec} \quad (1)$$

Completion time of a cloudlet (CT_c[][]) is determined by the sum of the execution time of the ith cloudlet on the jth VM and the ready time of the jth VM.

$$CT_c[i][j] = ET_c[i][j] + RT_{vm}[j]\ \text{measured in msec} \quad (2)$$

Total completion time of cloudlets (TCT_c) measures the completion time of all the cloudlets in the cloudlet bag. It measures the summation of completion time for all n cloudlets in their allocated VM among the m VMs based on the scheduling algorithm.

$TCT_c = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} CT_c[i][j]$ measured in msec (3)

Standard Deviation of Cloudlet completion time ($SDCD_{ct}[]$) measures the standard deviation for each cloudlet (cd_1, cd_2, ..., cd_n, where i=0 to n-1) to complete its execution across all VMs (vm_1, vm_2, ..., vm_m, where j=0 to m-1).

Mean Completion of cloudlet $MCT_c[i] = \frac{\sum_{j=0}^{m-1} CT_c[i][j]}{m}$ (4)

$SDCD_{ct}[i] = \sqrt{\frac{\sum_{j=0}^{m-1}(CT_c[i][j] - MCT_c[i])^2}{m}}$ (5)

Makespan or Schedule Length (MS_{SL}) represents the maximum completion time of all n cloudlets.

Makespan $MS_{SL} = \text{Maximum}(CT_c[i][j])$ measured in msec, where $MS_{SL} \leq TCT_c$ (6)

Average Virtual Machine utilization ratio (AVM_{ur}) refers to the average percentage of VMs being utilized in executing a set of cloudlets over a specific period of time. It provides an indication of how effectively the VMs are utilized in processing the assigned cloudlets and the overall efficiency of VM utilization in the scheduling process. A higher average VM utilization ratio indicates that the VMs are effectively utilized, while a lower ratio suggests potential underutilization of the VMs.

$AVM_{ur} = \frac{\sum_{j=0}^{m-1} CT_{vm}[j]}{MS_{SL} * m}$ (7)

Degree of Load Imbalance (DL_{IB}) is a metric used to quantify the extent of an uneven or imbalanced distribution of computational workload or cloudlets among VMs. A higher value of DL_{IB} indicates a greater degree of imbalance, suggesting that certain VMs are handling huge workloads compared to other VMs. Conversely, a lower DL_{IB} value indicates a more balanced distribution of workload across VMs, leading to better resource utilization and system performance. It is particularly important to achieve fairness or equity in resource allocation, ensuring that no VM is overloaded while others are underutilized.

$DL_{IB} = m * \frac{(\text{Maximum}(CT_{vm}[j] - \text{Minimum}(CT_{vm}[j])))}{\sum_{j=0}^{m-1} CT_{vm}[j]}$ (8)

Standard Deviation of Virtual Machine completion time ($SDVM_{ct}$) is used to measure the spread or variability of completion times across different VMs. A higher $SDVM_{ct}$ indicates greater variability in completion times among VMs, while a lower $SDVM_{ct}$ suggests more consistency in completion times across the VMs. $SDVM_{ct}$ is important to achieve consistent performance or minimize variability in cloudlet completion times across VMs.

$MVM_{ct} = \frac{\sum_{j=0}^{m-1} CT_{vm}[j]}{m}$ measured in msec (9)

$SDVM_{ct} = \sqrt{\frac{\sum_{j=0}^{m-1}(CT_{vm}[j] - MVM_{ct})^2}{m}}$ measured in msec (10)

Heterogeneity Index among cloudlets ($HetInx_{cd}$) assists in evaluating the diversity among cloudlets within the cloudlet bag in terms of their characteristics and resource requirements. AHICS utilizes cloudlet length as a primary measure to determine the heterogeneity index of the cloudlets. The $HetInx_{cd}$ is based on the co-variance of cloudlet length.

Mean Cloudlet Length $MCL = \frac{\sum_{i=0}^{n-1} \text{Cloudlet}[i].\text{Length}}{n}$ measured in MI (11)

Standard Deviation of Cloudlet Length $SD_{CDL} = \sqrt{\frac{\sum_{i=0}^{n-1}(\text{Cloudlet}[i].\text{Length} - MCL)^2}{n}}$ measured in MI (12)

$HetInx_{cd} = \frac{SD_{CDL}}{MCL} * 100$ (13)

Heterogeneity Index among Virtual Machines ($HetInx_{VM}$) assists in evaluating the diversity among VM characteristics, capacities, and configurations, such as processing speed, memory, storage capacity, network bandwidth, and other relevant attributes. AHICS makes use of processing speed, RAM, and bandwidth limits to measure the VM heterogeneity index. This heterogeneity index is a combined index based on the heterogeneity among the processing speed, RAM, and bandwidth size of the VMs. However, the measure for the heterogeneity index of VM that is based on the speed alone is depicted here. The other two are estimations are done in a similar manner. The $HetInx_{vm}$ is based on the co-variance of the virtual machines speed.

Mean Virtual Machine Speed $MVMS = \frac{\sum_{j=0}^{m-1} VM[j].\text{Speed}}{m}$ measured in MIPS` (14)

Standard Deviation of Virtual Machine Speed $SD_{VMS} = \sqrt{\frac{\sum_{i=0}^{n-1}(VM[j].\text{Speed} - MVMS)^2}{m}}$ measured in MIPS

$HetInx_{vm} = \frac{SD_{VMS}}{MVMS} * 100$ (15)

Combined Heterogeneity Index (CHetInx) measures the combined heterogeneity index of both the cloudlets and the VMs.

$CHetInx = \frac{HetInx_{cd} + HetInx_{vm}}{2}$ (16)

4. PROPOSED ALGORITHMS

4.1 Variance Managed Heuristic Scheduler (VMHS)

The VMHS operates in four phases, as outlined below. The scheduler computes the completion time $CT_c[i][j]$ for each cloudlet on each VM. It then computes the standard deviation of completion time for each cloudlet across different VMs. The algorithm identifies the cloudlet with the highest standard deviation of completion time and the VM with the earliest completion time for that specific cloudlet. It then maps the selected cloudlet to the identified VM and updates the ready time $RT_{vm}[j]$ of the VM. The process repeats until all cloudlets are scheduled. A cloudlet with a higher standard deviation of completion time among VMs indicates significant variability in its completion times when assigned to different VMs. Allocating these cloudlets to the VM with the earliest completion time aims to balance the workload distribution and reduce the load imbalance. The proposed VMHS algorithm demonstrates improved performance in environments characterized by low heterogeneity indexes. However, the VMHS algorithm produces mixed performance in scenarios where the heterogeneity index is high.

Algorithm I – Variance Managed Heuristic Scheduler (VMHS)

```
Input:
A list of n cloudlets with varying lengths measured in million instructions (MI):
List <Cloudlets> CDL:=($cd_1$, $cd_2$, $cd_3$… $cd_n$), where i | $1 \leq i \leq n$
A list of m virtual machines with varying processing power measured in million instructions per second (MIPS):
List <Virtual Machines> VML:=($vm_1$, $vm_2$, $vm_3$ … $vm_m$), where j | $1 \leq j \leq m$
Given Configuration (n, m)
Output:
A schedule with minimized makespan, improved VM utilization ratio, reduced load imbalance, and reduced standard
deviation of completion time among the VMs:
CloudletVmSchedule: = returnMap<Cloudlet, VirtualMachine>
Computational Matrices:
Matrix   1: $ET_c[i][j]$: Execution time of cloudlet $cd_i$ on virtual machine $vm_j$
Matrix   2: $RT_{vm}[j]$: Ready time of virtual machine $vm_j$
Matrix   3: $CT_c[i]\ [j]$: Completion time of cloudlet $cd_i$ on virtual machine $vm_j$
Matrix   4: $SDCD_{ct}[i]$: Standard deviation of cloudlet completion time for each ($cd_1$, $cd_2$, $cd_3$ … $cd_n$) across virtual
            machines ($vm_1$, vm2, vm3…$vm_m$)
Begin VMHS-Scheduler (Configuration n, m)
Phase 1: Compute $CT_c[n][m]$ matrix
1.  For each cloudlet $cd_i$ in the cloudlet list CDL:
2.  For each virtual machine $vm_j$ in the virtual machine list VML:
3.  Compute $CT_c[i][j]:=ET_c[i][j]+ RT_{vm}[j]$
4.  End loop of step 2
5.  End loop of step 1
Phase 2: Compute $SDCD_{ct}[n]$ matrix
6.  For each cloudlet $cd_i$ in the cloudlet list CDL:
7.  For each virtual machine $vm_j$ in the virtual machine list VML:
8.  Compute $SDCD_{ct}[i]$
9.  End loop of step 7
10. End loop of step 6
Phase 3: Selection of cloudlet and virtual machine
11. Find a cloudlet $cd_i$ with highest $SDCD_{ct}[i]$ value
12. Find a virtual machine $vm_j$ with earliest completion time CTc[i][j] for cloudlet $cd_i$
Phase 4: Mapping cloudlet to the virtual machine
13. Map the cloudlet $cd_i$ to the virtual machine $vm_j$
14. Discard the cloudlet $cd_i$ from the cloudlet list CDL
15. Update the ready time $RT_{vm}[j]$ of the virtual machine $vm_j$
16. Do until there is no cloudlet left in the cloudlet list CDL, repeat steps 1-15
End VMHS-Scheduler
```

Time Complexity Analysis of VMHS

The time complexity analysis of the VMHS scheduler is conducted in four phases. In Phase I, the completion time matrix, with dimensions n x m, is computed, resulting in a time complexity of O(n * m). Similarly, Phase II computes the standard deviation of completion time, also taking O(n * m) time. In Phase III, the cloudlet with the highest standard deviation of completion time and the virtual machine with the earliest finishing time are determined, incurring time complexities of O(n) and O(m), respectively. Thus, the total time complexity for Phase III is O(n + m). In the final phase, cloudlet-to-virtual-machine mapping and updating the ready time of virtual machines are performed, both in constant time. The algorithm iterates until no cloudlets remain in the cloudlet bag, which takes O(n) time. The time complexity is calculated as O(n * m) + O(n * m) + O(n + m) + O(n). Since the dominant time complexity is O(n * m), the overall time complexity simplifies to O(n * m).

4.2 MaxMin Scheduler

MaxMin cloudlet scheduling is a well-established heuristic algorithm known for minimizing makespan compared to other heuristic scheduling algorithms. This algorithm sorts cloudlets by their length and assigns the largest cloudlet to the VM with the earliest finishing time. It prioritizes large-sized cloudlets to minimize its completion time there by ensuring reduced makespan, efficient VM utilization, and load balancing. This optimization is attributed to the fact that smaller-sized cloudlets typically require fewer computational resources and less execution time than larger ones. Additionally, MaxMin benefits smaller cloudlets by leaving enough resources for their utilization. The MaxMin algorithm performs well, especially in scenarios where smaller cloudlets outnumber the larger ones. The MaxMin algorithm yields better results in scenarios where the heterogeneity index is high. In some cases, the MaxMin algorithm suffers in performance, specifically in low heterogeneity scenarios.

Algorithm II – MaxMin Scheduler

```
Begin MaxMin-Scheduler (Configuration)
Phase 1: Compute $CT_c[n][m]$ matrix
1.  For each cloudlet $cd_i$ in the cloudlet list CDL:
2.  For each virtual machine $vm_j$ in the virtual machine list VML:
3.  Compute $CT_c[i][j]:=ET_c[i][j]+ RT_{vm}[j]$
4.  End loop of step 2
5.  End loop of step 1
Phase 2: Selection of cloudlet and virtual machine
6.  Find the minimum completion time $CT_c[i][j]$ for all cloudlets
7.  Find a cloudlet $cd_i$ with maximum value among the minimum completion time $CT_c[i][j]$
8.  Find a virtual machine $vm_j$ with earliest completion time $CT_c[i][j]$ for cloudlet $cd_i$
Phase 3: Mapping cloudlet to the virtual machine
9.  Map the cloudlet $cd_i$ to the virtual machine $vm_j$
10. Discard the cloudlet $cd_i$ from the cloudlet list CDL
11. Update the ready time $RT_{vm}[j]$ of the virtual machine $vm_j$
12. Do until there is no cloudlet left in the cloudlet list CDL, repeat steps 1-11
End MaxMin-Scheduler
```

4.3 Adaptive Heterogeneity Index Cloudlet Scheduler (AHICS)

VMHS is well-suited for low-heterogeneity environments where cloudlets and VMs have similar characteristics, allowing for effective workload balancing and consistent performance. MaxMin, on the other hand, is designed to handle diverse workloads in high-heterogeneity environments, ensuring fair resource allocation and preventing VM overload. AHICS is proposed to address the limitations of VMHS and MaxMin in heterogeneous cloud environments. The AHICS algorithm is designed to be adaptive for all heterogeneity levels. AHICS considers the CHetInx among the cloudlets and VMs in order to select either VMHS or the MaxMin scheduler. When the heterogeneity index is low, AHICS schedules the cloudlets using VMHS. It chooses the MaxMin scheduler when the heterogeneity index is high. Experimental evaluation consistently shows that AHICS performs well in both cases, achieving its objectives. AHICS begins by determining the $HetInx_{cd}$, the $HetInx_{vm}$, and the CHetInx. If the CHetInx is low, it calls the VMHS scheduler; otherwise, it invokes the MaxMin scheduler to map the cloudlets and the VMs. If the CHetInx is less than or equal to 25, the VMHS scheduler is selected. Otherwise, the MaxMin scheduler is invoked to manage allocation.

Algorithm III – Adaptive Heterogeneity Index Cloudlet Scheduler (AHICS)

```
Input:
Given Configuration (n, m)
n: Number of cloudlets, where $i \mid 1 \leq i \leq n$
m: Number of virtual  machines, $j \mid 1 \leq j \leq m$
Output:
A schedule with minimized makespan, improved VM utilization ratio, reduced load imbalance, and reduced standard
deviation of completion time among the VMs:
CloudletVmSchedule: = returnMap<Cloudlet, VirtualMachine>
Assessment Metrics: $MS_{SL}$, $AVM_{ur}$, $DL_{IB}$, $SDVM_{ct}$
Computational Matrices:
Matrix 1: CLen[n]: Stores cloudlet length
Matrix 2: VMSpeed[m]: Stores virtual machine speed
Begin AHICS-Scheduler (Configuration)
Phase 1: Compute Heterogeneity Index
$HetInx_{cd}$:=Calculate-Covcd (CLen[n])
$HetInx_{vm}$:=Calculate-Covvm (VMSpeed[m])
CHetInx:=Mean($HetInx_{cd}$ + $HetInx_{vm}$)
```

```
Phase 2: Check CHetInx and select the scheduler
If (CHetInx<=25) then
Invoke VMHS-Scheduler
CloudletVmSchedule:=VMHS-Scheduler(Configuration)
Else
Invoke MaxMin-Scheduler
CloudletVmSchedule:=MaxMin-Scheduler(Configuration)
End If
Phase 3: Measure performance metrics
$MS_{SL}$:=MeasureMakespan(CloudletVmSchedule)
$AVM_{ur}$:=MeasureVmUtilization(CloudletVmSchedule)
$DL_{IB}$:=MeasureDLIB(CloudletVmSchedule)
$SDVM_{ct}$:=MeasureSDVM(CloudletVmSchedule)
Phase 4: Ascertain ($MS_{SL}$, $AVM_{ur}$, $DL_{IB}$, $SDVM_{ct}$)
End AHICS-Scheduler
```

Time Complexity Analysis of AHICS

Phase I involves calculating the Coefficient of Variation (CoV) for cloudlets and virtual machines by iterating over the cloudlet lengths (n) and virtual machine speeds (m), respectively. Computing the mean of CoV involves a constant-time operation. Therefore, the time complexity of Phase I is O(n + m). Phase II involves a simple conditional check based on the computed heterogeneity index. The time complexity of this phase is O(1) since it involves constant-time operations. In phase III, various performance metrics are measured, which involves iterating over the generated schedule. These measurements have a combined time complexity of O(n * m). Ascertaining the performance metrics evaluates the overall performance of the scheduler. The time complexity of this phase is O(1). Overall, the time complexity is O(n + m) + O(1) + O(n * m) + O(1), resulting in a time complexity of O(n * m).

5. EXPERIMENTAL SETUP

This experimental evaluation utilizes the CloudSim 3.0.3 simulator, a Java-based framework widely used for simulating cloud infrastructures and services. Cloudlets and VMs with diverse configurations and numbers are employed to evaluate the performance of the proposed AHICS algorithm. The analysis utilizes cloudlets ranging from 10 to 1500 and VMs ranging from 3 to 32 numbers with different specifications or configurations. The cloudlets used in this study were synthetic cloudlets generated using CloudSim 3.0.3.

These cloudlets were designed to represent various types of cloud applications. A simulated environment with varying numbers and configurations of VMs was also created programmatically. Cloudlets ranging in size from 1000 MI to 25016250 MI and VMs ranging from 100 MIPS to 625000 MIPS were evaluated. The experimental setup, including all configurations, is shown in Table 4.

Table 4: Cloudlet and VM configuration

Components	*Parameters*	*Parameterization*
Datacenter	Datacenter Instances	1
	Host Instances	12
	VM scheduler policy	VmSchedulerSpaceShared
	Cores or processing units	Dual/Quad
	Bandwidth	10000
	Operating System	Linux
	Hypervisor (VMM)	Xen
	Host memory capacity in MB	100000
Virtual Machines	VM count	3-32
	VM speed in MIPS	10000-14000
	Processing Elements PEs	1
	VM memory in MB	512
	Bandwidth	1000
	Cloudlet scheduler policy	CloudletSchedulerSpaceShared
Cloudlets	Cloudlet length in MI	36000-95000
	Cloudlet count	10-1500
	Processing Elements PEs	1

The time complexity of the proposed VMHS, MaxMin and AHICS schedulers along with the time complexity of the comparison algorithms are shown in Table 5.

Table 5: **Time Complexity Table**

Algorithms	*MinMin*	*MaxMin*	*TASA*	*HAMM*	*PTFR*	*RSSM*	*VMHS*	*AHICS*
Time Complexity	$O(mn^2)$	$O(mn^2)$	$O(mn^2)$	$O(mn^2)$	$O(mn^2)$	$O(n^2 m \log m)$	$O(n * m)$	$O(n * m)$ or $O(mn^2)$

The flowchart of the proposed AHICS scheduler is illustrated in Figure 3. It depicts the selection of scheduling algorithm based on CHetInx for mapping cloudlets to the VMs during different scenarios.

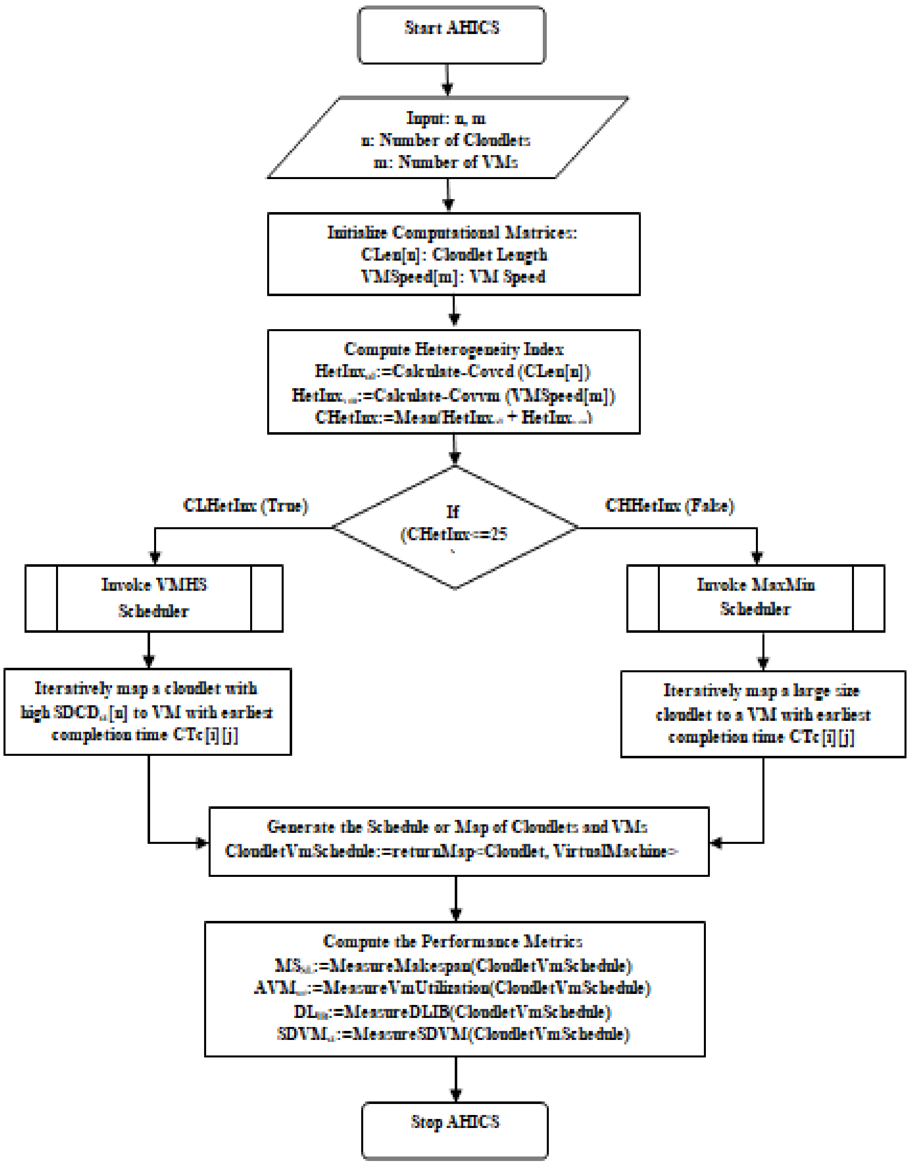

Start AHICS
Input: n, m
n: Number of Cloudlets
m: Number of VMs
Initialize Computational Matrices:
CLen[n]: Cloudlet Length
VMSpeed[m]: VM Speed
Compute Heterogeneity Index
HetInx_cl:=Calculate-Covcd (CLen[n])
HetInx_vm:=Calculate-Covvm (VMSpeed[m])
CHetInx:=Mean(HetInx_cl + HetInx_vm)
CLHetInx (True)
If (CHetInx<=25
CHHetInx (False)
Invoke VMHS Scheduler
Invoke MaxMin Scheduler
Iteratively map a cloudlet with high SDCD_cl[n] to VM with earliest completion time CTc[i][j]
Iteratively map a large size cloudlet to a VM with earliest completion time CTc[i][j]
Generate the Schedule or Map of Cloudlets and VMs
CloudletVmSchedule:=returnMap<Cloudlet, VirtualMachine>
Compute the Performance Metrics
MS_AL:=MeasureMakespan(CloudletVmSchedule)
AVM_ut:=MeasureVmUtilization(CloudletVmSchedule)
DL_IB:=MeasureDLIB(CloudletVmSchedule)
SDVM_cl:=MeasureSDVM(CloudletVmSchedule)
Stop AHICS

Figure 3: **Flow Chart of AHICS Scheduler**

5.1 Illustration

For illustration purposes, a configuration comprising 10 cloudlets and 4 VMs is considered, with two varying levels of CHetInx. The CHetInx score is calculated based on the cloudlets varying lengths (measured in Millions of Instructions, MI) and the VMs processing speeds (measured in Million Instructions per Second, MIPS). The performance of the algorithms, assessed in terms of makespan, is depicted in Table 6. The specific configurations of these cloudlets and VMs are detailed below:

Case I: CLHetInx configuration

CDL[]={69000,67000,66000,62000,65000,60000,64000,68000,63000,61000}

VML[]={3500,4500,5000,4000}

$$\left[\begin{array}{ccccc} & \mathrm{VM0} & \mathrm{VM1} & \mathrm{VM2} & \mathrm{VM3} \\ \mathrm{CD0} & 19.71 & 15.33 & 13.8 & 17.25 \\ \mathrm{CD1} & 19.14 & 14.89 & 13.40 & 16.75 \\ \mathrm{CD2} & 18.86 & 14.67 & 13.20 & 16.50 \\ \mathrm{CD3} & 17.71 & 13.78 & 12.40 & 15.50 \\ \mathrm{CD4} & 18.57 & 14.44 & 13.00 & 16.25 \\ \mathrm{CD5} & 17.14 & 13.33 & 12.00 & 15.00 \\ \mathrm{CD6} & 18.29 & 14.22 & 12.80 & 16.00 \\ \mathrm{CD7} & 19.43 & 15.11 & 13.60 & 17.00 \\ \mathrm{CD8} & 18.00 & 14.00 & 12.60 & 15.75 \\ \mathrm{CD9} & 17.43 & 13.56 & 12.20 & 15.25 \end{array}\right] \qquad \left[\begin{array}{c} SD \\ 2.2093 \\ 2.1451 \\ 2.1148 \\ 1.9843 \\ 2.0823 \\ 1.9215 \\ 2.0520 \\ 2.1790 \\ 2.0182 \\ 1.9540 \end{array}\right]$$

For the CLHetInx schedule, the VMHS scheduler is used. The heterogeneity index for this schedule is 8.80. The completion time and standard deviation matrices from iteration one shown above indicate that cloudlet CD0 has the highest standard deviation, and VM2 is the VM with the earliest finishing time. Therefore, cloudlet CD0 is allocated to VM2 as dictated by the VMHS scheduler. The entire schedule is depicted below.

Case I schedule: {CD5->1, CD0->2, CD9->3, CD3->0, CD8->1, CD7->2, CD6->3, CD4->0, CD1->2, CD2->1}

Case II: CHHetInx configuration

CDL[]={125000,35000,55000,200000,450000,320000,95000,75700,550000,600000}

VM[]={13500,2500,15000,4000}

$$\left[\begin{array}{ccccc} & \mathrm{VM0} & \mathrm{VM1} & \mathrm{VM2} & \mathrm{VM3} \\ \mathrm{CD0} & 044.44 & 240.00 & 040.00 & 150.00 \\ \mathrm{CD1} & 040.74 & 220.00 & 036.67 & 137.50 \\ \mathrm{CD2} & 033.33 & 180.00 & 030.00 & 112.50 \\ \mathrm{CD3} & 023.70 & 128.00 & 021.33 & 080.00 \\ \mathrm{CD4} & 014.81 & 080.00 & 013.33 & 050.00 \\ \mathrm{CD5} & 009.26 & 050.00 & 008.33 & 031.25 \\ \mathrm{CD6} & 007.04 & 038.00 & 006.33 & 023.75 \\ \mathrm{CD7} & 005.61 & 030.28 & 005.05 & 018.93 \\ \mathrm{CD8} & 004.07 & 022.00 & 003.67 & 013.75 \\ \mathrm{CD9} & 002.59 & 014.00 & 002.33 & 008.75 \end{array}\right]$$

For the CHHetInx schedule, the MaxMin scheduler is used. The heterogeneity index for this schedule is 72.43. The completion time matrix of iteration one shown above indicate that cloudlet CD9 is the largest, and VM2 is the VM with the earliest finishing time. Therefore, cloudlet CD9 is allocated to VM2 as dictated by the MaxMin scheduler. The entire schedule is depicted below.

Case II schedule: {CD9->2, CD8->0, CD0->1, CD3->3, CD5->0, CD7->3, CD4->2, CD6->0, CD2->1, CD1->2}. Graphs 1 and 2 depict the makespan for the two aforementioned schedules: the schedule with a low heterogeneity index (CLHetInx) and the schedule with a high heterogeneity index (CHHetInx).

Table 6: Makespan for Illustration

Algorithm	*Makespan (MS_{SL}) in msec*	
	CLHetInx=8.80	*CHHetInx=72.32*
MinMin	43.43	103.8
TASA	43.43	76.77
HAMM	42.77	72.43
PTFR	43.43	95.77
RSSM	43.21	79.36
AHICS	42.10	72.43

Figure 1: Makespan comparison for CLHetInx

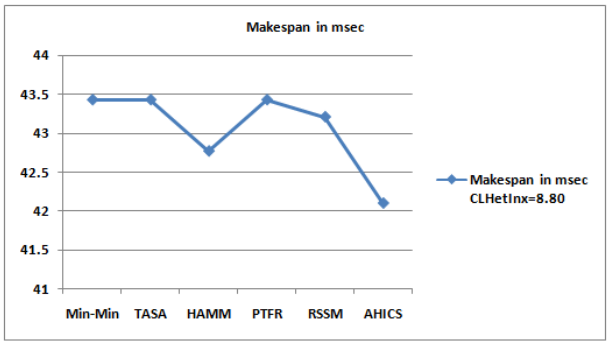
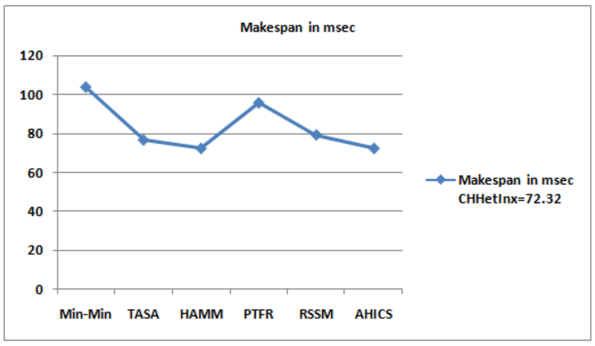


Figure 2: Makespan comparison for CHHetInx



6. RESULTS AND DISCUSSION

The performance of the proposed AHICS algorithm is compared against several established heuristic algorithms, including MinMin, TASA, HAMM, PTFR, and RSSM. These algorithms were selected for comparison due to their similar heuristic nature and better performance compared to other heuristic algorithms in the literature. Given their demonstrated effectiveness, these algorithms were taken for comparison with the proposed AHICS scheduling algorithm [38-40]. A total of 75 simulations were conducted with different cloudlet and VM configurations. Of these, 40 representative trials are listed in Table 7. Table 7 presents the makespan (**MS_{SL}**) performance of AHICS compared to the other schedulers. Tables 8 to 10 show the effectiveness of AHICS in terms of VM utilization ratio (**AVM_{ur}**), degree of load imbalance (**DL_{IB}**), and standard deviation of VM completion time (**$SDVM_{ct}$**). AHICS demonstrates enhanced performance across multiple metrics, such as makespan, VM utilization, degree of load imbalance, and the standard deviation of completion time among VMs.

Table 7: Performance Measure for MS_{SL}

S. No.	*Configuration (n, m, CHetInx)*			*Makespan (MS_{SL}) in msec*					
	n	*m*	*CHetInx*	*Min-Min*	*TASA*	*HAMM*	*PTFR*	*RSSM*	*AHICS*
1	177	17	10.38	472.09	472.09	472.09	472.1	469.35	466.99
2	200	20	31.4	378.79	375.36	362.94	376.25	365.21	362.94
3	201	15	10.28	611.95	611.95	611.95	611.93	611.02	609.2
4	231	19	13.16	791.89	791.89	791.89	791.88	794.4	788.89
5	251	25	56.18	569.83	569.83	569.83	571.62	546.44	533.37
6	300	30	42.4	569.29	554.77	543.46	571.46	551.22	543.46
7	305	16	15.83	132.38	132.38	132.38	132.34	131.47	131.28
8	310	16	45.34	942.01	926.37	901.57	942.59	912.9	901.57
9	316	16	10.35	2708.38	2708.38	2708.38	2708.18	2705.94	2696.31
10	381	17	12.42	1105.17	1105.17	1105.17	1105.19	1099.12	1097.08
11	413	17	36.38	1652.85	1652.85	1652.85	1672.1	1620.51	1620.24
12	421	17	12.95	1396.24	1396.24	1396.24	1396.3	1394.95	1382.29
13	437	19	19.27	2013.33	2013.33	2013.33	2014.14	2003.95	1993.64
14	456	19	11.81	1961.54	1951.51	1950.26	1961.06	1948.54	1944.31
15	464	20	53.77	6020.1	5965.72	5815.22	6012.43	5901.2	5815.22
16	480	22	20.01	1751.79	1743.32	1731.51	1752.65	1739.41	1731.51
17	540	21	19.9	1821.1	1806.4	1806.4	1820.12	1809.48	1805.66
18	600	24	54.44	7371.73	7281.34	7079.2	7308.06	7192.38	7079.2
19	650	25	21.79	17961.17	17852.22	17815.1	17981.51	17788.81	17706.18
20	700	27	20.65	1609.5	1602.77	1592.97	1609.72	1590.81	1592.8
21	750	28	30.42	949.57	944.15	934.98	948.33	936.13	934.98

22	801	24	15.35	5709.72	5709.72	5709.72	5711.21	5689.05	5687.33
23	875	26	44.85	3901.18	3901.18	3901.18	3909.94	3833.45	3803.03
24	935	28	45.13	5399.95	5399.95	5399.95	5391	5324.51	5267.96
25	985	29	50	973.37	973.37	973.37	970.33	956.48	948.69
26	1000	30	43.46	435.8	431.47	424.53	436.18	426.27	424.53
27	1050	28	7.34	4920.27	4922.92	4918.56	4920.27	4924.02	4917.36
28	1075	29	55.27	15257.32	15257.32	15257.32	15191.14	15059.51	14869.12
29	1100	30	56.79	1776.26	1762.95	1735.27	1785.47	1747.23	1735.27
30	1130	31	58.68	1688.27	1673.08	1645.57	1680.42	1660.93	1645.13
31	1159	33	39.2	11292.1	11292.1	11292.1	11266.26	11124.32	11096.6
32	1185	33	13.17	9612.91	9612.91	9612.91	9612.8	9584.69	9584.02
33	1200	34	36.52	9801.31	9757.74	9659.95	9787.55	9687.31	9659.95
34	1250	35	13.56	12817.58	12810.78	12794.72	12815.32	12796.67	12782.76
35	1250	36	50.81	5169.5	5141.16	5037.36	5167.72	5074.24	5037.36
36	1285	37	48.99	11070.89	11070.89	11070.89	11109.99	10915.06	10809.4
37	1310	38	50.51	11031.25	10920.33	10698.12	10999.15	10821.05	10698.12
38	1375	39	4.55	7619.91	7619.91	7619.91	7619.91	7618.34	7613.32
39	1450	40	40.71	7146.22	7092.18	7037.85	7143.26	7063.87	7037.85
40	1500	41	6.7	5082.77	5079.24	5080.33	5082.76	5080.14	5077.73

Table 8: Performance Measure for AVM_{ur}

S. No.	***Configuration (n, m, CHetInx)***			***Average Virtual Machine utilization ratio (AVM_{ur})***					
	n	***m***	***CHetInx***	***MinMin***	***TASA***	***HAMM***	***PTFR***	***RSSM***	***AHICS***
1	1050	28	7.34	0.9871	0.9861	0.9874	0.9871	0.9859	0.9876
2	1185	33	13.17	0.985	0.985	0.985	0.9851	0.9882	0.9881
3	1250	35	13.56	0.9848	0.9857	0.9868	0.985	0.9867	0.9876
4	1375	39	4.55	0.9861	0.9861	0.9861	0.9861	0.9861	0.9868
5	1500	41	6.7	0.9864	0.9872	0.9869	0.9864	0.987	0.9876
6	1200	34	36.52	0.9727	0.9783	0.9911	0.9733	0.9868	0.9911
7	1250	36	50.81	0.961	0.9727	0.999	0.9629	0.9888	0.999
8	1285	37	48.99	0.967	0.967	0.967	0.9645	0.9875	0.9992
9	1310	38	50.51	0.9548	0.974	0.9995	0.9613	0.984	0.9995
10	1450	40	40.71	0.9677	0.9808	0.991	0.9685	0.9862	0.991

Table 9: Performance Measure for DL_{IB}

S. No.	***Configuration (n, m, CHetInx)***			***Degree of Load Imbalance (DL_{IB})***					
	n	***m***	***CHetInx***	***MinMin***	***TASA***	***HAMM***	***PTFR***	***RSSM***	***AHICS***
1	1050	28	7.34	0.0271	0.0335	0.0266	0.0271	0.0329	0.0275
2	1185	33	13.17	0.0379	0.0379	0.0379	0.0376	0.0291	0.0308
3	1250	35	13.56	0.039	0.0349	0.0355	0.0386	0.033	0.0313
4	1375	39	4.55	0.0286	0.0286	0.0286	0.0286	0.0289	0.0299
5	1500	41	6.7	0.0311	0.0303	0.0298	0.0311	0.0322	0.0281
6	1200	34	36.52	0.1043	0.0782	0.0495	0.1096	0.0469	0.0495

7	1250	36	50.81	0.1814	0.1086	0.0036	0.2315	0.046	0.0036
8	1285	37	48.99	0.1327	0.1327	0.1327	0.1468	0.0265	0.003
9	1310	38	50.51	0.2426	0.1402	0.0021	0.2323	0.053	0.0021
10	1450	40	40.71	0.1879	0.1267	0.0538	0.1673	0.0788	0.0538

Table 10: Performance Measure for $SDVM_{ct}$

S. No.	***Configuration (n, m, CHetInx)***			***Standard Deviation of Virtual Machine completion time ($SDVM_{ct}$) in msec***					
	n	***m***	***CHetInx***	***MinMin***	***TASA***	***HAMM***	***PTFR***	***RSSM***	***AHICS***
1	1050	28	7.34	38.6535	40.9218	37.6535	38.6535	40.6168	37.9364
2	1185	33	13.17	92.1933	92.1933	92.1933	91.7104	75.03	76.8312
3	1250	35	13.56	130.2131	111.5994	104.6436	129.7273	104.1401	103.38
4	1375	39	4.55	64.0872	64.0872	64.0872	64.0881	59.5385	61.2843
5	1500	41	6.7	42.75	43.9576	40.736	42.7474	43.9518	39.3524
6	1200	34	36.52	235.5662	171.3314	89.7608	251.3577	111.7352	89.7608
7	1250	36	50.81	212.4574	110.8042	5.0402	214.4849	44.7738	5.0402
8	1285	37	48.99	357.2741	357.2741	357.2741	351.9609	76.769	6.8925
9	1310	38	50.51	512.5018	276.128	5.5658	446.9224	154.9083	5.5658
10	1450	40	40.71	263.1339	161.4544	74.9475	241.2531	101.9347	74.9475

Following this detailed analysis, Tables 11 and 12 present the average values of the aforementioned metrics, summarizing the overall performance of AHICS across both low (CLHetInx) and high (CHHetInx) combined heterogeneity index (CHetInx) cases. AHICS demonstrates enhanced performance across multiple metrics, such as makespan, VM utilization, degree of load imbalance, and the standard deviation of completion time among VMs. Table 13 shows the percentage improvement of AHICS compared to the other schedulers.

Table 11: Performance Measures for CLHetInx Configurations

S. No.	***Metrics***	***Combined Low Heterogeneity Index (CLHetInx) Configurations***					
		Min-Min	***TASA***	***HAMM***	***PTFR***	***RSSM***	***AHICS***
1	**MS_{SL}**	3390.686	3383.886	3380.627	3391.59	3376.846	3269.646
2	**AVM_{UR}**	0.930968	0.967176	0.9686	0.965324	0.972636	0.97616
3	**DL_{IB}**	0.083044	0.082812	0.076288	0.08228	0.065228	0.062752
4	**$SDVM_{ct}$**	45.45103	42.59807	38.61237	44.46759	34.49734	34.11433

Table 12: Performance Measures for CHHetInx Configurations

S. No.	***Metrics***	***Combined High Heterogeneity Index (CHHetInx) Configurations***					
		Min-Min	***TASA***	***HAMM***	***PTFR***	***RSSM***	***AHICS***
1	**MS_{SL}**	4352.858	4334.504	4296.309	4345.759	4281.727	4145.666
2	**AVM_{UR}**	0.935588	0.945724	0.964224	0.938524	0.975252	0.988124
3	**DL_{IB}**	0.247868	0.236768	0.128712	0.255096	0.118004	0.041548
4	**$SDVM_{ct}$**	198.7718	169.3082	97.89007	182.9269	68.47412	17.65003

Reducing makespan is essential for faster cloudlet completion and minimizing waiting time. A smaller makespan allows the cloud system to process more tasks per unit of time, thereby increasing throughput. In time-sensitive applications like video streaming or online gaming, a low makespan is crucial to avoid delays. Additionally, reducing makespan can lead to more efficient resource utilization, reducing costs, and improving performance and user experience. Graphs 3 and 4, which depict the improved makespan (MS_{SL}) performance of AHICS in both low and high combined heterogeneity cases (CHetInx), highlight its potential in cloud environments with diversity.

Table 13: **Consolidated Performance Measures**

Schedulers	*Makespan (MS_{SL})*		*Average VM utilization ratio (AVM_{ur})*		*Degree of Load Imbalance (DL_{IB})*		*Standard Deviation of VM completion time*	
	CLHetInx()	*CHHetInx*	*CLHetInx*	*CHHetInx*	*CLHetInx*	*CHHetInx*	*CHHetInx*	*CLHetInx*
MinMin	3.7	5	5.85	5.62	24.44	83.24	24.94	91.12
TASA	3.49	4.56	1.93	4.48	24.22	82.45	19.92	89.58
HAMM	3.39	3.63	1.78	2.48	17.74	67.72	11.65	81.97
PTFR	3.73	4.83	1.12	5.28	23.73	83.71	23.28	90.35
RSSM	3.28	3.28	1.36	1.32	3.8	64.79	1.11	74.22

Figure 3: **Makespan for CLHetInx**

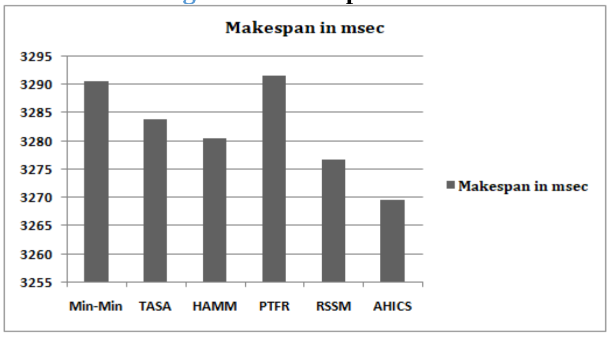
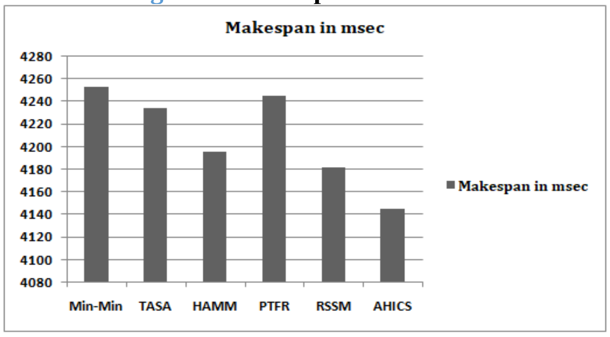


Figure 4: **Makespan for CHHetInx**



Similarly, Graphs 5 and 6 depict the improved performance of AHICS in terms of VM utilization ratio (AVM_{ur}). Improved VM utilization results in reduced cost and energy consumption.

Figure 5: **AVM_{ur} for CLHetInx**

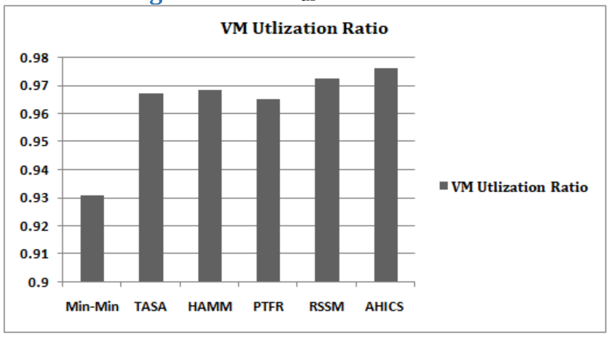
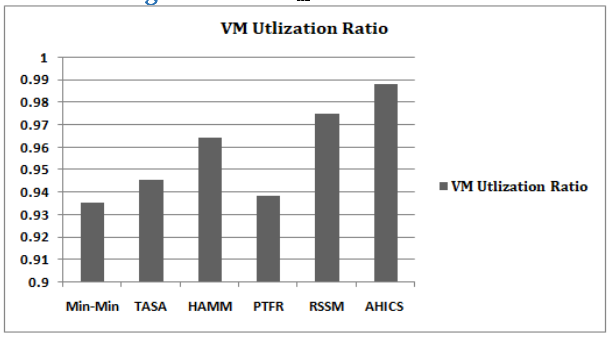


Figure 6: **AVM_{ur} for CHHetInx**



Graph 7 and 8 represents the performance of AHICS in measure of degree of load imbalance (DL_{IB}). It ensures that cloudlets are distributed evenly across VMs, preventing bottlenecks and optimizing resource utilization. A balanced distribution of the workload helps maintain consistent performance for applications running in the cloud.

Figure 7: **DL_{IB} for CLHetInx**

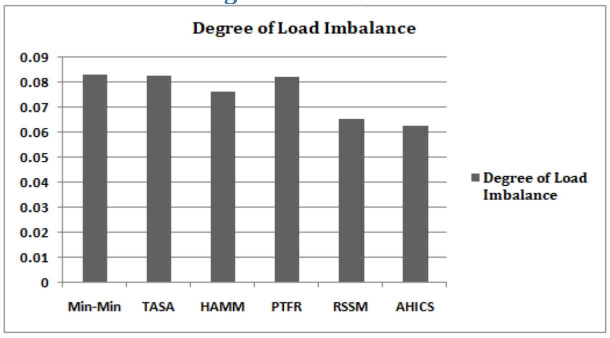
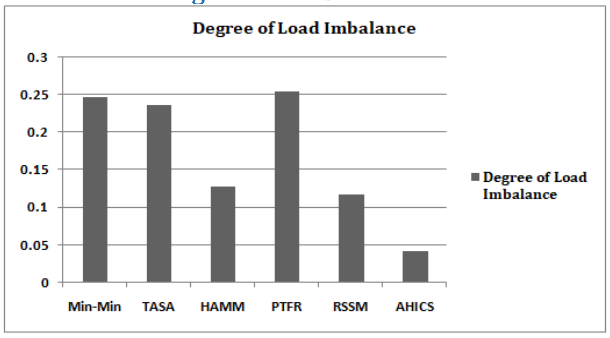


Figure 8: **DL_{IB} for CHHetInx**



Graphs 9 and 10 depict the improved performance of AHICS in terms of standard deviation of completion time among VMs ($SDVM_{ct}$). It represents fairness in the treatment of VMs, with a smaller deviation indicating that all VMs are treated more equally.

The experimental results show that AHICS generally outperforms MinMin, TASA, HAMM, PTFR, and RSSM. While HAMM exhibit similar performance to AHICS in terms of makespan, AHICS consistently achieves better results in all simulation runs. The RSSM produces a nearly equal makespan only when the number of VMs is odd.

Figure 9: **SDVM$_{ct}$ for CLHetInx**

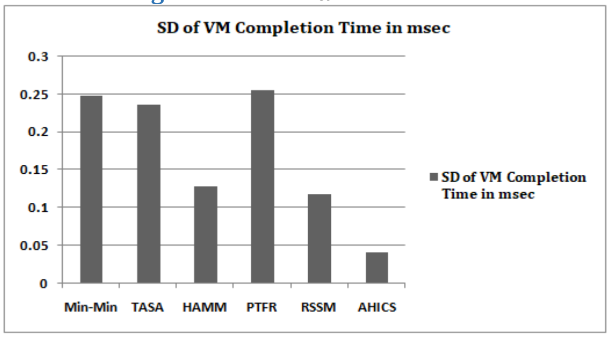
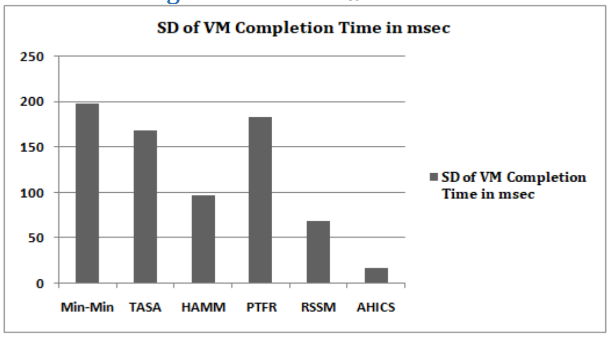


Figure 10: **SDVM$_{ct}$ for CHHetInx**



6.1 Statistical t-Test

Pair-wise comparisons of the algorithms were conducted using a statistical t-test to evaluate the performance of the baseline AHICS algorithm against other algorithms. In a t-Test, a high t-value and a low p-value generally indicate a statistically significant difference between two groups. The statistical t-Test result is depicted in Table 13, with the t-value and p-value. The results demonstrate that AHICS produces an improved makespan compared to the others. AHICS consistently outperformed MinMin, TASA, HAMM, PTFR, and RSSM, as evidenced by the statistically significant t-Test results (MinMin: t=3.579, p=0.006; TASA: t=3.481, p=0.007; PTFR: t=3.626, p=0.006; RSSM: t=3.135, p=0.012). While there was no significant difference between AHICS and HAMM (t=1.121, p=0.291), AHICS consistently achieved a lower makespan than HAMM in all simulation runs. In real-time applications like gaming, financial transactions, or tasks with strict deadlines, even a few milliseconds of latency can be detrimental, and even small improvement in makespan can have a significant impact. AHICS additionally excelled in other three metrics.

Table 13: **Performance of AHICS based Statistical t-Test**

Treatment 1 Algorithm	*Treatment 2 Algorithm*	*Mean Difference between pairs*	*Sum of Squares*	*t-statistics*	*p-value*
MinMin	AHICS	122.795	782,195,066.406	3.579	0.006
TASA	AHICS	79.063	779,348,718.769	3.481	0.007
HAMM	AHICS	12.132	777,358,234.702	1.121	0.291
PTFR	AHICS	119.365	782,245,224.142	3.626	0.006
RSSM	AHICS	40.568	777,230,430.891	3.135	0.012

7. CONCLUSION

The heterogeneity among cloudlets and VMs significantly impacts scheduling and overall cloud performance. This heterogeneity issue has not been addressed effectively in the literature. AHICS, a novel three-tier scheduling algorithm, effectively addresses this heterogeneity issues. AHICS dynamically selects between VMHS and MaxMin schedulers based on the CHetInx. AHICS consistently outperforms other heuristic schedulers like MinMin, TASA, HAMM, PTFR, and RSSM. All critical metrics considered were significantly improved. The reduction in makespan is crucial for real-time cloud services as it results in reduced latency, improved response time, and throughput. The VM utilization ratio is crucial for cost savings and energy efficiency. Achieving a balanced workload distribution optimizes resource utilization, cost, and energy consumption. Minimizing the variation in completion time among VMs ensures fairness in resource utilization. This evaluation demonstrates the effectiveness and adaptability of AHICS in heterogeneous environments. While HAMM exhibit nearly comparable performance in terms of makespan, AHICS consistently achieves better results in all four measures, notably with lower time complexity. While AHICS demonstrates improved performance, cloud environments are inherently diverse, with varying workloads demands, and priorities. Some applications prioritize minimizing response time, while others focus on minimizing latency, maximizing throughput, or optimizing VM utilization, etc.

To enhance its adaptability and performance, integrating ML and DQL within AHICS can be beneficial. By creating a dynamic heterogeneity estimator using ML, we can predict future heterogeneity levels based on historical data. This allows AHICS to proactively adjust its scheduling strategy, optimizing performance. DQL can learn optimal scheduling policies directly from experience. By interacting with the cloud environment, DQL can discover the best actions (VMHS or MaxMin) for different states (heterogeneity). Both ML and DQL can continuously learn and adapt to changing conditions. Adapting DQL ensures that AHICS remains effective in even dynamic cloud environments.

DECLARATION STATEMENT

Ethical Statement

I will conduct myself with integrity, fidelity, and honesty. I will openly take responsibility for my actions, and only make agreements, which I intend to keep. I will not intentionally engage in or participate in any form of malicious harm to another person or animal.

Informed Consent for data Used

All subjects gave their informed consent for inclusion before they participated in the study. The study was conducted in accordance with the Declaration of Helsinki.

I consent to participate in the research project and the following has been explained to me: the research may not be of direct benefit to me. My participation is completely voluntary, my right to withdraw from the study at any time without any implications to me.

Data Availability

- Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.
- The datasets used and/or analyzed during the current study are available from the corresponding author on reasonable request.
- All data generated or analyzed during this study are included in this published article

CONFLICT OF INTEREST

The authors declare that they have no conflict of interest.

Competing Interests

The authors have no competing interests to declare that are relevant to the content of this article.

FUNDING DETAILS

No funding was received to assist with the preparation of this manuscript.

Authors Contributions

The author **D Gritto** involved in design, and theoretical formalism, performed the analytic calculations and performed the numerical simulations and evaluation process presented in the paper. The author **P Muthulakshmi** helped to derive the mathematical equation and background study of the paper and also provided a factual review and helped to edit the manuscript.

ACKNOWLEDGEMENTS

I am grateful to all of those with whom I have had the pleasure to work during this and other related Research Work. Each of the members of my Dissertation Committee has provided me extensive personal and professional guidance and taught me a great deal about both scientific research and life in general.

REFERENCES

[1]. Z. N. Al-kateeb and D. B. Abdullah, "Unlocking the Potential: Synergizing IoT, Cloud Computing, and Big Data for a Bright Future," *Iraqi Journal for Computer Science and Mathematics*, vol. 5, no. 3, pp. 1-13, 2024.

[2]. C. C. A. Vieira, et al., "RAaaS: Resource Allocation as a Service in multiple cloud providers," *Journal of Network and Computer Applications*, vol. 221, p. 103790, 2024.

[3]. S. Ghafir, et al., "Load balancing in cloud computing via intelligent PSO-based feedback controller," *Sustainable Computing: Informatics and Systems*, vol. 41, p. 100948, 2024.

[4]. T. Deepa and D. Cheelu, "A comparative study of static and dynamic load balancing algorithms in cloud computing," in 2017 *International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, 2017.

[5]. I. Behera and S. Sobhanayak, "Task scheduling optimization in heterogeneous cloud computing environments: A hybrid GA-GWO approach," *Journal of Parallel and Distributed Computing*, vol. 183, p. 104766, 2024.

[6]. H. Mikram, S. El Kafhali, and Y. Saadi, "HEPGA: A new effective hybrid algorithm for scientific workflow scheduling in cloud computing environment," *Simulation Modelling Practice and Theory*, vol. 130, 2024.

[7]. N. R. Sabat, et al., "Hybrid technique for optimal task scheduling in cloud computing environments," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 22, no. 2, pp. 380-392, 2024.

[8]. B. Rambabu, A. V. Reddy, and S. Janakiraman, "Hybrid artificial bee colony and monarchy butterfly optimization algorithm (HABC-MBOA)-based cluster head selection for WSNs," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 5, pp. 1895-1905, 2022.

[9]. K. L. Devi and S. Valli, "Multi-objective heuristics algorithm for dynamic resource scheduling in the cloud computing environment," *The Journal of Supercomputing*, vol. 77, no. 8, pp. 8252-8280, 2021.

[10]. A. A. Nasr, et al., "Cloudlet scheduling based load balancing on virtual machines in cloud computing environment," *Journal of Internet Technology*, vol. 20, no. 5, pp. 1371-1378, 2019.

[11]. K. Kamalam, et al., "Scheduling tasks in cloud setup using pair-task heuristic," in AIP Conference Proceedings, vol. 2901, no. 1, *AIP Publishing*, 2023.

[12]. N. M. Reda, et al., "Range-Suffrage Algorithm for Grid Task," *International Journal of Applied and Physical Sciences*, vol. 1, no. 2, pp. 42-50, 2015.

[13]. N. M. Reda, et al., "Sort-Mid tasks scheduling algorithm in grid computing," *Journal of Advanced Research*, vol. 6, no. 6, pp. 987-993, 2015.

[14]. S. M. Khamis, N. M. Reda, and W. Zakaria, "An improved Sort-Mid algorithm for scheduling heterogeneous grid tasks," *Journal of Supercomputing*, vol. 78, no. 2, pp. 3072-3090, 2022.

[15]. I. Syed, "HAMM: A hybrid algorithm of Min-Min and Max-Min task scheduling algorithms in cloud computing," *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 9, pp. 209-218, 2020.

[16]. J. Y. Maipan-Uku, et al., "An extended min-min scheduling algorithm in cloud computing," i-manager’s *Journal on Cloud Computing*, vol. 5, no. 2, pp. 20, 2018.

[17]. M. Kumar and S. C. Sharma, "Deadline constrained based dynamic load balancing algorithm with elasticity in cloud environment," *Computers & Electrical Engineering*, vol. 69, pp. 395-411, 2018.

[18]. J. P. B. Mapetu, Z. Chen, and L. Kong, "Heuristic cloudlet allocation approach based on optimal completion time and earliest finish time," *IEEE Access*, vol. 6, pp. 61714-61727, 2018.

[19]. A. R. Khan, "Dynamic Load Balancing in Cloud Computing: Optimized RL-Based Clustering with Multi-Objective Optimized Task Scheduling," *Processes*, vol. 12, no. 3, p. 519, 2024.

[20]. Z. Tong, et al., "DDMTS: A novel dynamic load balancing scheduling scheme under SLA constraints in cloud computing," *Journal of Parallel and Distributed Computing*, vol. 149, pp. 138-148, 2021.

[21]. U. K. Jena, P. K. Das, and M. R. Kabat, "Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 6, pp. 2332-2342, 2022.

[22]. M. Adil, et al., "CA-MLBS: content-aware machine learning based load balancing scheduler in the cloud environment," *Expert Systems*, vol. 40, no. 4, p. e13150, 2023.

[23]. Elsakaan and K. Amroun, "A novel multi-level hybrid load balancing and tasks scheduling algorithm for cloud computing environment," *The Journal of Supercomputing*, pp. 1-41, 2024.

[24]. R. Vijay and T. R. Sree, "Resource Scheduling and Load Balancing Algorithms in Cloud Computing," *Procedia Computer Science*, vol. 230, pp. 326-336, 2023.

[25]. R. N. Hanuman, et al., "Virtual Machine Load Balancing Using Improved ABC for Task Scheduling in Cloud Computing," in International Conference on Intelligent Computing and Networking, Singapore: *Springer Nature* Singapore, 2023.

[26]. M. S. Al Reshan, et al., "A fast converging and globally optimized approach for load balancing in cloud computing," *IEEE Access*, vol. 11, pp. 11390-11404, 2023.

[27]. S. T. Milan, et al., "Priority-based task scheduling method over cloudlet using a swarm intelligence algorithm," *Cluster Computing*, vol. 23, no. 2, pp. 663-671, 2020.

[28]. D. A. Shafiq, et al., "A load balancing algorithm for the data centres to optimize cloud computing applications," *IEEE Access*, vol. 9, pp. 41731-41744, 2021.

[29]. Z. Ahmed, A. F. Ashrafi, and M. Mahbub, "Clustering based Max-Min Scheduling in Cloud Environment," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 9, 2017.

[30]. S. T. Dehkordi and V. K. Bardsiri, "TASA: A new task scheduling algorithm in cloud computing," *Journal of Advances in Computer Engineering and Technology*, vol. 1, no. 4, pp. 25-32, 2015.

[31]. S. Sethi, A. Sahu, and S. K. Jena, "Efficient load balancing in cloud computing using fuzzy logic," *IOSR Journal of Engineering*, vol. 2, no. 7, pp. 65-71, 2012.

[32]. K. Ramya and S. Ayothi, "Hybrid dingo and whale optimization algorithm-based optimal load balancing for cloud computing environment," *Transactions on Emerging Telecommunications Technologies*, vol. 34, no. 5, p. e4760, 2023.

[33]. M. Gamal, et al., "Osmotic bio-inspired load balancing algorithm in cloud computing," *IEEE Access*, vol. 7, 2019.

[34]. H. Singh, S. Tyagi, and P. Kumar, "Cloud resource mapping through crow search inspired metaheuristic load balancing technique," *Computers & Electrical Engineering*, vol. 93, p. 107221, 2021.

[35]. A. Chraibi, S. Ben Alla, and A. Ezzati, "Makespan optimisation in cloudlet scheduling with improved DQN algorithm in cloud computing," *Scientific Programming*, vol. 2021, pp. 1-11, 2021.

[36]. A. Joshi and S. D. Munisamy, "Evaluating the performance of load balancing algorithm for heterogeneous cloudlets using HDDB algorithm," *International Journal of System Assurance Engineering and Management*, vol. 13, Suppl 1, pp. 778-786, 2022.

[37]. P. Kumar, M. Bundele, and D. Somwansi, "An adaptive approach for load balancing in cloud computing using load balancing," in 2018 *3rd International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE)*, 2018.

[38] E. Aarthi, S. Jagan, C. P. Devi, J. J. Gracewell, S. B. Choubey, A. Choubey, et al., "A turbulent flow optimized deep fused ensemble model (TFO-DFE) for sentiment analysis using social corpus data," Social Network Analysis and Mining, vol. 14, p. 41, 2024.

[39] A. Mahalingam, G. Perumal, G. Subburayalu, M. Albathan, A. Altameem, R. S. Almakki, et al., "ROAST-IoT: a novel range-optimized attention convolutional scattered technique for intrusion detection in IoT networks," Sensors, vol. 23, p. 8044, 2023.

[40] M. S. Sheela, S. Gopalakrishnan, I. P. Begum, J. J. Hephzipah, M. Gopianand, and D. Harika, "Enhancing Energy Efficiency With Smart Building Energy Management System Using Machine Learning and IOT," Babylonian Journal of Machine Learning, vol. 2024, pp. 80-88, 2024.