

SLA-DE-RALBA: Cost-efficient dynamic enhanced resource-aware load balancing algorithm for cloud computing

Mohsin Nawaz¹, Altaf Hussain¹, Marran Al Qwaid², and Muazzam A. Khan Khattak³

¹ Department of Computer Science, KICSIT Campus, Institute of Space Technology, Islamabad 44000, Pakistan

mohsin_mscs@kicsit.edu.pk, altaf.hussain@ist.edu.pk (corresponding author)

² Department of Computer Science, College of Computing and Information Technology, Shaqra University, Saudi Arabia
maldossari@su.edu.sa

³ Computer Science Department, Quaid i Azam University, Islamabad, Pakistan
muazzam.khattak@qau.edu.pk

Abstract. Cloud computing has become a dominant paradigm for delivering scalable and flexible on-demand resources; however, efficiently executing high performance computing (HPC) workloads remains challenging, particularly in heterogeneous environments. Conventional static scheduling methods often lead to poor resource utilization and increased makespan, while dynamic approaches improve load distribution but introduce significant overhead due to continuous monitoring and real-time decision-making. To address these challenges, this paper proposes an SLA-aware Dynamic Enhanced Resource-Aware Load Balancing Algorithm (SLA-DE-RALBA). The algorithm minimizes load imbalance by considering the computational capacities of virtual machines and ensures Service Level Agreement (SLA) compliance through a three-tier priority-based workflow. The proposed approach is evaluated using CloudSim Plus on two benchmark datasets: Heterogeneous Computing Scheduling Problem (HCSP) instances and the Google Cloud Jobs dataset. Results demonstrate that SLA-DE-RALBA consistently outperforms baseline algorithms, including RALBA, DRALBA, DE-RALBA, SLA-RALBA, Dynamic Max-Min, PSSLB, and PSSELB, across key metrics such as makespan, resource utilization, job rejection, throughput, execution time, and cost. Notably, it achieves zero job rejection, reduces energy consumption by up to 85%, improves resource utilization by 11.9%, lowers makespan by 41–45%, and decreases execution time by up to 57%, making it a robust and efficient solution for HPC workload scheduling in cloud environments.

Keywords: Cloud Computing, HPC Workload Scheduling, SLA-Aware Load Balancing, Dynamic Resource Management, Heterogeneous Cloud Environments.

1. Introduction

The National Institute of Standards and Technology (NIST) defines cloud computing as a paradigm that facilitates ubiquitous, convenient, and on-demand network access to a

common pool of configurable computing resources which can be provisioned and discharged with speed having little management overheads and limited service provider intervention [1]. With such a paradigm, it can be easily get in touch with the computational servers, software frameworks, storage disks and also networking systems which may be procured and installed without much interruption on the operational process [2,3]. Previous research on the heuristics of scheduling and resource efficiency address intrinsic benefits of cloud environments such as cost efficiency, elasticity and operational flexibility since cloud resources are available all over the world through the use of the internet [4]. Moreover, cloud computing enables quick implementation of services and applications as well as improves coordination and scalability in dispersed computing settings [5]. Task scheduling is still a core issue in cloud computing since it involves the effective mapping of computational tasks onto accessible resources (e.g. virtual machines (VMs) and physical servers) and also the optimum use of resources, system performance and at times conflicting, goals, including response time, throughput, energy efficiency and adherence to service level agreement [6,7,8]. It is especially pronounced in heterogeneous cloud environments, where dynamic changes in workload and diversity of resources often result in uneven load distribution and under-utilization of resources [9]. From an architectural view, the scheduling decisions may be classified into the two inter-related layers: job-to-VM mapping which assigns the tasks to the virtual machines depending on resource availability and the SLA-related restrictions and VM-to-physical machine mapping which also ensures optimal usage of the hardware facilities by reducing both energy and performance degradation [10].

Modern studies are progressively being transitioned to a SLA-consciousness scheduling strategy with both heuristic and metaheuristic models and other methods that dynamically respond to changing loads and system states. Such approaches mainly aim at maximizing system performance and minimizing the time to complete the tasks (makespan) of the system to achieve maximum energy efficiency and ensuring resiliency and robustness SLA satisfaction despite the dynamic and uncertain conditions related to cloud operating conditions [11]. Dynamic scheduling makes it possible to redistribute workloads dynamically by varying the decisions regarding task allocation according to the current system states and hence offers greater capabilities to handle dynamic and unexpected execution situations [12]. Scheduling strategies may also be categorized into online and offline approaches: online scheduling takes a step-by-step decision based on the arrival of the tasks, while offline scheduling is based on full information about all jobs and aims at globally optimal allocation schemes. Besides there can be preemptive and non-preemptive scheduling policies as far as the ability to interrupt tasks and restart at a later time during the execution can be present or absent [13]. The recent development around cloud scheduling has brought advanced heuristic and meta-heuristic methods, including the Cuckoo Optimization based scheduling that seeks to reduce resource wastage and time of task completion (makespan) [14]. Also new studies focus on green and energy-efficient time-conscious scheduling systems to minimize resource utilization, and fault tolerant and adaptive energy-conscious schedulers that openly trade goals of performance with sustainability demands [15]. Cloud service providers often use approaches of over-provisioning as a mitigation approach to Service Level Agreement (SLA) violations, negatively impacting resource usage, workload imbalance and influencing the presence of tasks and in turn prolonging makespan and reducing provider revenues. To overcome these shortcom-

ings, SLA-RALBA [16] came up with SLA-aware extension of the Resource-Aware Load Balancing Algorithm (RALBA) by the representation of three different categories of SLA which include time-centric, balanced and cost-centric and their combination in terms of the execution cost and execution time in the scheduling process.

The scheduling mechanism used by SLA-RALBA in order to enhance utilization across the virtual machines is based on two phases i.e. Fill and Spill phases [10]. Experimental assessments of both GoCJ and HCSP work loads modeling using CloudSim resulted in demonstrating that SLA-RALBA delivered better performance compared to existing dynamic and proactive scheduling heuristics such as PSSLB, PSSELB, Dynamic Maximum Minimization and DRALBA based strategies in terms of resource utilization, makespan and load balance [16]. However, in the light of such developments a common scheduling model which concurrently conducts both SLA awareness and dynamic run-time flexibility is yet to be well studied [18]. Current strategies are used to trade-off one or two objectives in isolation leading to natural trade-offs that restrict the total efficiencies of the entire system especially at the machine-level load balancing level as is evidenced in earlier research [9,14]. Based on this requirement, SLA-DE-RALBA is designed to combine SLA-based scheduling with run-time flexibility to deliver a balanced trade-off between performance and cost and cloud environments. The major implications of the given research are as follows:

1. A detailed literature review identified key limitations in existing static and dynamic cloud scheduling approaches regarding SLA compliance, flexibility, cost, job rejection, and energy efficiency.
2. A major research gap was identified where SLA-aware algorithms lack dynamic adaptability, while dynamic approaches fail to satisfy SLA requirements with better resource utilization in cloud environments.
3. The proposed SLA-DE-RALBA algorithm integrates SLA-awareness, dynamic resource optimization, and energy-efficient heuristics to improve throughput, resource utilization, job acceptance, cost efficiency, and energy savings.
4. Experimental evaluation using the CloudSim Plus simulator with GoCJ and HCSP datasets demonstrated that SLA-DE-RALBA outperformed several existing cloud scheduling algorithms.

2. Related Work

Cloud job scheduling has been given much attention by the academia and industry since its direct influence on system throughput, resource usage, energy consumption and user satisfaction [7]. Various solutions have been proposed at both ends of the spanning of classical heuristics and multi-criteria SLA-awareness and dynamic scheduling. This paper does not provide a comprehensive survey of scheduling methods; instead, it highlights well-known algorithms representing key advances in cloud scheduling research.

Several dynamic scheduling approaches have been proposed to improve runtime adaptability. Dynamic Min-Max [36] reduces waiting time, improves response time, and increases throughput through concurrent VM allocation, while ECMM [26] enhances load balancing and response time using runtime monitoring of tasks and VMs. Although effective for dynamic workload management, these approaches do not explicitly consider

SLA constraints or cost-awareness [29]. PSSLB and related proactive scheduling approaches within the RePro-Active framework [25] use simulation-based predictions and hybrid Round Robin strategies to optimize task allocation. They improve makespan, resource utilization, and throughput by assigning tasks based on estimated VM performance, while PSSELB further enhances parallelism by prioritizing longer tasks. However, both methods incur additional simulation overhead and do not rely on prior workload knowledge, limiting scalability in large cloud environments. Albalawi [13] proposed a hybrid dynamic scheduling framework combining multiple metaheuristics (RRA-SWO, HAGA, LRT, and LR-HSA) for load balancing in cloud systems. Evaluated on CloudSim with 51 VMs, it achieves high task success, throughput, and resource utilization. However, it lacks SLA-aware differentiation, energy efficiency considerations, and validation in large-scale multi-cloud environments. Hagrass and El-Sayed [37] proposed Balanced Independent-Task Assignment (BITA) and its hard real-time counterpart BITAr for fault-tolerant scheduling of independent tasks in heterogeneous clouds, prioritizing tasks based on deadline urgency. Evaluated on GoCJ, the methods outperform FTTA and RADL in utilization and task rejection with low computational overhead. However, they do not consider SLA-aware differentiation or execution cost. Zhu [38] proposed Load Balancing using Ant Colony and Whale Optimization (LBAW), a hybrid ACO-WOA framework for task scheduling, resource optimization, and VM migration in cloud environments. It improves utilization, makespan, and response time over existing methods, as shown in MATLAB experiments on the San Diego dataset. However, it introduces high computational overhead and complex parameter tuning, limiting scalability in large dynamic systems. Beena et al. [38] proposed an energy-aware cloud framework combining Kubernetes with ACO, PSO, and DVFS for resource scaling and container management. It improves energy efficiency and performance, but suffers from ACO-induced overhead and limited real-world validation. RALBA was proposed to address the limitations of traditional scheduling methods such as Min-Min and Round Robin, which often cause VM overload and inefficient resource utilization [19]. It employs a two-phase strategy: the fill phase allocates tasks based on VM processing power, while the spill phase assigns remaining tasks to the VM with the earliest completion time [17]. Experimental results showed significant reductions in makespan and improved resource utilization [10]. DRALBA [27] extends RALBA with a Dynamic-RALBA scheduler and Dynamic Updater for real-time VM load management, making it effective for compute-intensive tasks in heterogeneous environments. CloudSim evaluations with HCSP, GoCJ, and synthetic workloads show high resource utilization, reduced makespan, and improved throughput over several existing methods. However, it does not address energy-aware optimization or multi-cloud interoperability. SLA-RALBA [16] extends RALBA by incorporating cost-aware and SLA-driven scheduling decisions, supporting time-based, cost-based, and hybrid SLA models. It retains the Fill and Spill scheduling framework while aligning task allocation with user SLA requirements [22]. Experimental evaluation on the CloudSim platform demonstrated improved makespan and resource utilization compared to several baseline algorithms. However, SLA-RALBA follows a static scheduling approach, limiting its adaptability to dynamic workload changes during execution [12]. To overcome the limited flexibility of SLA-RALBA, DE-RALBA [23] was proposed with real-time VM state updates and dynamic workload redistribution. It improves adaptability to workload variations, reduces idle time, and minimizes performance bottlenecks through con-

tinuous VM monitoring and rescheduling. Table 1 summarizes the reviewed scheduling algorithms, including their techniques, methodologies, strengths, and weaknesses.

3. Proposed SLA-DE-RALBA Algorithm

The section provides an introduction to the baseline algorithms and a detailed demonstration of the proposed SLA-DE-RALBA algorithm including its architectural framework, system modeling mechanism, performance evaluation model and algorithm code.

SLA-DE-RALBA is a hierarchical architecture as shown in Figure 1, and has five different levels that are used to manage the resources and schedule cloud tasks. The core

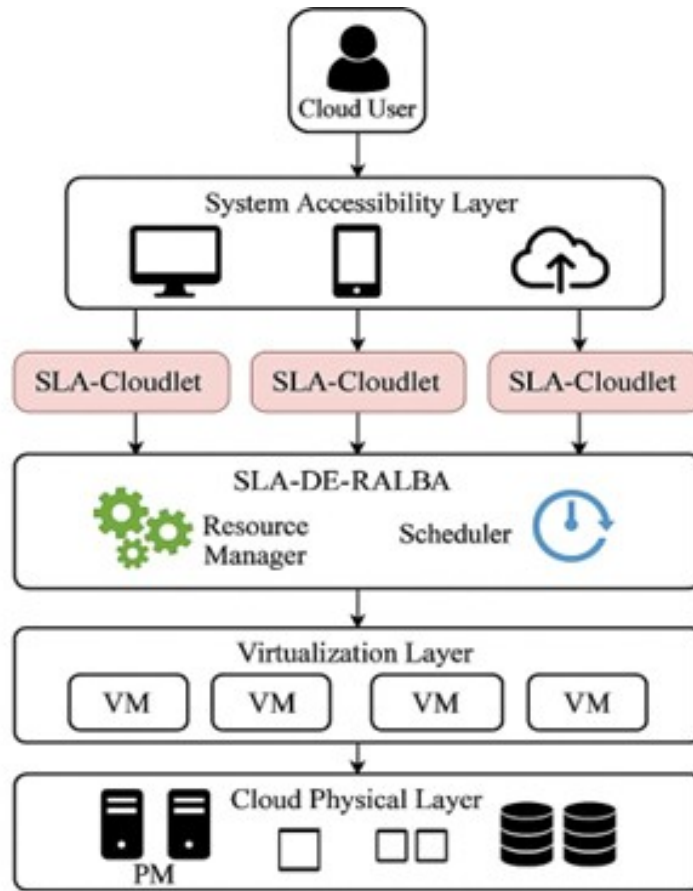


Fig. 1. Abstract Level Diagram

layer is the algorithmic nucleus of SLA-DE-RALBA which consists of a Resource Manager, Scheduler, and Load Balancer. The Resource Manager computes VM shares using workload, SLA weights, and utilization, and enables priority tasks to use idle resources.

Table 1. Comparative Analysis of Existing Scheduling Algorithms

Algorithm	Methodology	Strengths	Weaknesses
RALBA [31]	Two-phase scheduling. The Fill scheduler assigns tasks based on VM computational power; the Spill scheduler allocates remaining tasks to the VM with the earliest completion time.	Reduces makespan vs. Min-Min. Achieves over 7× better resource utilization than Random Selection; improves throughput.	Lacks SLA awareness. Static scheduling fails to adapt to runtime workload variations.
SLA-RALBA [16]	Extends RALBA by incorporating execution cost and supporting time-oriented, cost-oriented, and hybrid SLA models using Fill and Spill schedulers.	Improves makespan and resource utilization while maintaining cost efficiency through balanced time-cost trade-offs.	Higher computational overhead due to cost calculations; less effective for computation-intensive workloads.
DE-RALBA [23]	Employs dual schedulers: one for VM assignment based on processing power and earliest finish time; another to dynamically update VM status.	Achieves higher ARUR and lower makespan on GoC1 and HCSP datasets. Enhances throughput for dynamic workloads.	Runtime monitoring introduces overhead and increases system complexity in large-scale environments.
DRALBA [27]	Combines Dynamic-RALBA scheduling with a Dynamic-Updater to continuously adjust VM loads and computation shares during execution.	Improves resource utilization, reduces makespan, and enhances throughput and scalability.	Runtime updates incur overhead; lacks energy-awareness and multi-cloud support.
Dynamic [36]	Real-time scheduling assigns tasks immediately based on estimated execution time without batch sorting.	Improves throughput, efficiency, and load balancing while reducing waiting time in dynamic environments.	Performance declines with high resource heterogeneity; ignores SLA and cost constraints.
PSSLB [25]	Simulation-based approach that sorts tasks in descending order and assigns them to balance VM workloads using execution predictions.	Reduces task completion time, improves average resource utilization, and increases throughput.	Simulation overhead raises time complexity; prediction inaccuracies degrade performance for large datasets.
PSSSLB [25]	Enhances PSSLB through a pivot-task mechanism enabling parallel task allocation across heterogeneous VMs.	Delivers best throughput performance; improves makespan and ARUR over Min-Min variants.	High simulation and computation overhead; limited scalability under highly dynamic conditions.
Dynamic [26]	Enhanced Max-Min assigns the longest task to the VM with the shortest completion time using periodic updates.	Reduces waiting time and response ratio while improving load distribution and user satisfaction.	Periodic updates may miss real-time workload changes; can reduce resource utilization in some cases.
RRA-SWO [13]	Combines RR-SWO for allocation with HAGA scheduling and LRT-LR-HSA for load prediction.	Delivers high success rate, throughput, and efficient performance in CloudSim.	Does not address SLA compliance or energy efficiency.
BITA/BITAr [36]	Uses critically-based scheduling, extending from soft to hard real-time with FIFO and fault tolerance.	Delivers high utilization and low task rejection consistently across different workload sizes.	Does not consider SLA priorities or cost factors.
LBAW [37]	Combines ACO for task allocation with WOA for resource utilization and VM migration in a hybrid optimization approach.	Improves resource utilization, reduces makespan, imbalance and response time compared to existing methods.	Higher computational cost and parameter tuning complexity may limit performance in large, dynamic environments.
ARAA/ARKLM [38]	Integrates ACO, PSO, and DVFS in a Kubernetes framework for adaptive resource and container management.	Strong energy efficiency and scalability, with low CPU usage and fast execution performance.	High CPU usage of ACO and reliance on Kubernetes may limit wider adoption; real-world validation still limited.

The Scheduler assigns tasks via an SLA-based priority queue using multi-criteria VM scoring. The Load Balancer monitors VM variance and redistributes workloads when imbalance exceeds a threshold to prevent overload and ensure efficient resource use. The Cloud Physical Layer forms the underlying hardware infrastructure, including servers, storage, and networking resources that support virtualization. The system follows a bi-directional data flow where user requests move through SLA-based cloudlet abstraction, scheduling decisions, VM execution, and physical resource allocation. A continuous feedback loop via the VM Status Update mechanism (Algorithm 2) sends real-time VM status to the SLA-DE-RALBA core, enabling dynamic scheduling, resource reassignment, proactive load balancing, and sustained SLA compliance under varying workloads [32]. This study used a SLA mechanism with three layers; **Level 1** provides best-effort, lowest-priority service (weight 1.1), with tasks placed at the end of the queue and subject to preemption or suspension under resource contention. **Level 2** offers moderate priority service (weight 1.2) with balanced resource allocation, fair scheduling, and protection from starvation, maintaining a good cost–performance trade-off. **Level 3** delivers high-priority, performance-guaranteed service (weight 1.3), with preferential VM assignment, near-full resource access, and preemptive control over lower-priority tasks to ensure SLA compliance under heavy load.

3.1. System Architecture

The SLA-DE-RALBA architecture takes the entire multi-stage flow as a guideline in scheduling tasks in the clouds as depicted in Figure 2. It begins with cloudlets that have certain SLA, computational load (in Million Instructions) and time of arrival. The system then automatically shares dynamic VM in all the virtual machines according to computation ratios, total system workload, SLA weights and the current level of utilization. The suggested scheduling architecture functions on a two-way system of activation according to assessments executed on the initial capacity. In regular operation when the resources are adequate. The major pathway will take a simple chain of operations by initializing resources availability tabling of all virtual machine instances then ensuring entry points of incoming computational tasks. The scheduler then gives priorities to tasks as per service-level agreement requirement and allocates the most critical workload to the virtual machine with the most desirable performance properties and consistency within the system. On the other hand, once the total system capacity decrees beyond set points, the structure transitions to a different route which makes use of SLA-aware scheduling logic to locate significant pending tasks assign them to virtual machines with the best expected times to completion that meet service agreements and furthermore resource constraints and broadcast allocation changes throughout all monitoring systems.

These parallel execution paths are eventually combined at a strategic optimization level at which the scheduler is done to carry out detailed analysis of rescheduling opportunities. It entails the comparison of the recently estimated time of accomplishing the task with a possibly shorter time that could be completed at idle or lightly loaded virtual machines. When there is a chance of improving performance, the system is strategic in moving less important tasks out of the congested resources into the empty capacity thus improving the overall throughput and the efficiency in utilizing the resources used. Subsequent to this refinement process architecture starts load balancing processes in details

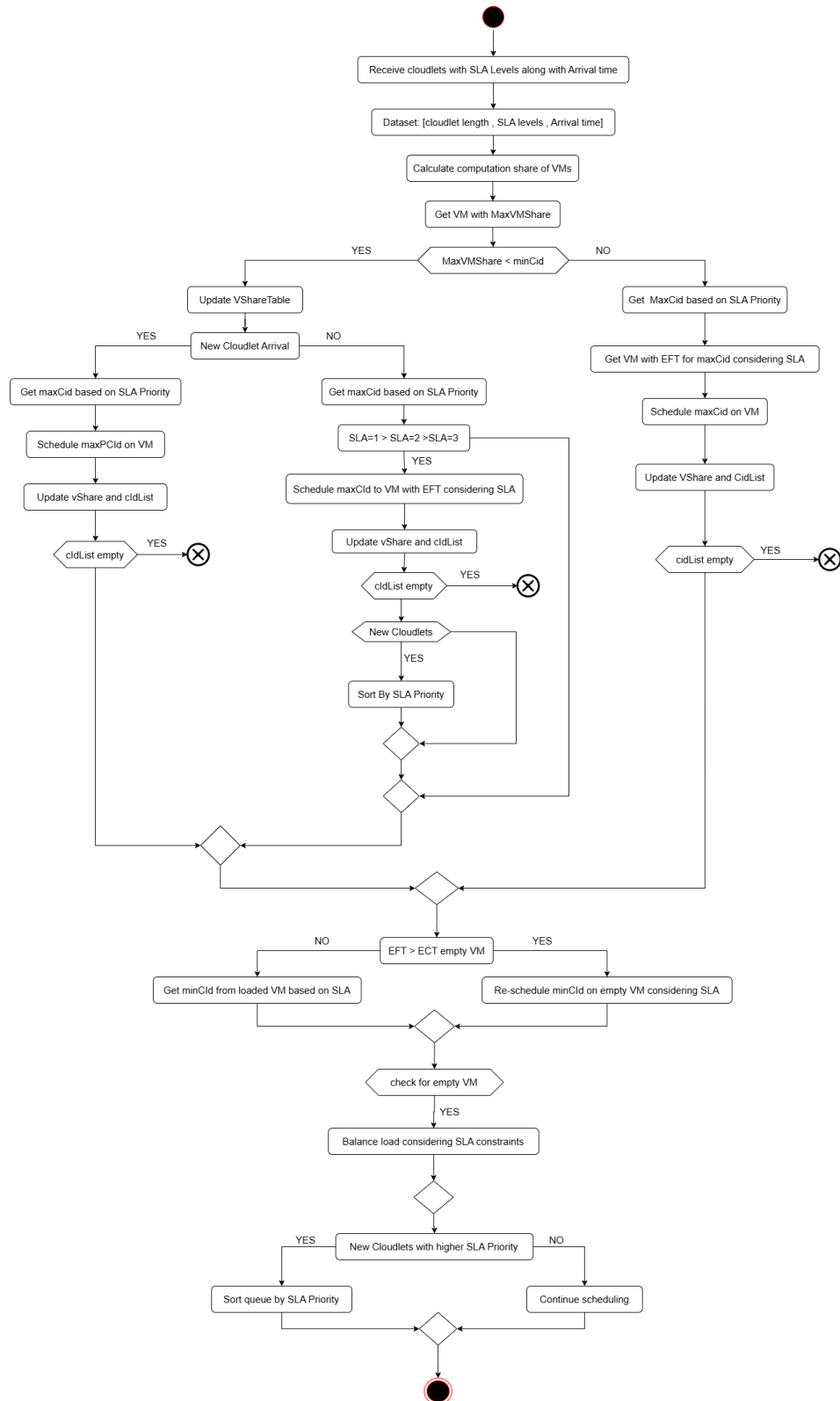


Fig. 2. Flow Diagram of SLA-DE-RALBA

where it is known that virtual machines are underutilized and calculates the variance measurements across the whole infrastructure. When there is more than configured levels of variance intelligent redistribution systems come into effect to offload computational workloads of saturated resources to unused capacity without notifying service-level guarantees during the migration period.

The framework includes advanced dynamic arrival processing work which constantly monitors the task submission pipeline specifically focusing on high-priority workloads of SLA Level 3. In case critical tasks are present in the system when running the task queue is dynamically restructured through extensive SLA-driven reorganization of all pending tasks to guarantee high-priority workloads a faster scheduling consideration as low-priority workloads are moved to their proper position. This real-time monitoring and responsiveness enables the SLA-DE-RALBA architecture to accommodate changing workload profiles ensure uniformity in compliance at all service layers automated resource mapping via smart virtual machine selection algorithms and dynamic capacity control solutions and stability in the long-run of the system through proactive intervening measures. All of these combined minimize the rate of task rejection, maximize the operational efficiency of a data center and provide differentiated service quality in accordance with hierarchical structures of service-level agreement.

3.2. System Model

The SLA-DE-RALBA model is a universal system of comprehensive management of the entire cloud scheduling process starting with the arrival of computing work and its ultimate execution. This architecture is developed on the basis of event-driven continuous architecture which incorporates four interdependent components service-level agreement conscious priority management, resource allocation mechanisms, flexible virtual machine choice plans and proactive load structure controls. The design brings together the architectural structure characterizing component interactions and the composition of operation specifications. On the sequential arrangement of the operations to schedule resource heterogeneity and the activities of redistributing loads across the entire simulation procedure. In addition various notations that were employed during this work are indicated in Table 2.

The **mathematical foundation** underlying the SLA-DE-RALBA algorithm facilitates intelligent task-to-virtual machine mapping decisions that enhance execution time minimization, system throughput maximization and balanced workload distribution.

$$\max \text{SLACId} = \max_{\forall p \in \text{SLACloudlets}} (\text{SLA Priority}(p) \times 10^6 + \text{Size}(p)) \quad (1)$$

where SLA Priority follows the hierarchy: $SLA-1 > SLA-2 > SLA-3$.

$\max \text{SLACId} = \max(\text{SLA Priority}(p) \times 10^6 + \text{Size}(p))$ Equation (1) determines the most critical task for processing by integrating service-level priority coefficients with computational workload dimensions to establish scheduling order.

$$\text{VMShare}_j = \text{vCRatio}_j \times \text{totalMI} \times \text{slaWeight}_j \times \text{utilizationFactor}_j \quad (2)$$

Equation (2) computes the adaptive virtual machine resource allocation proportion by incorporating processing capacity ratios, aggregate system workload, service-level weighting parameters, and current utilization metrics.

$$\text{slaWeight}_j = 1.0 + \frac{\text{avgSLA}_{\text{VM}_j}}{10.0} \quad (3)$$

Table 2. Notations used in SLA-DE-RALBA

Variable / Notation	Description
cld	Representation of user jobs on Cloud
vList	List of Virtual Machines in a datacenter
cldRepository	Repository of user tasks with arrival time and SLA levels
MIPS	Million Instructions Per Second (Computation Power of VM)
MI	Million Instructions (size of cld)
vCRatio	Computation ratio of all the VMs
maxPcld	Maximum possible cloudlet for a VM
vShare _i	Computation share of VM _i
newVShare	Modified computation share of a VM
vStatusTable	List that stores set of VM computation share, ready time, status of VM, and list of cloudlets associated to VM
vStatusTable-vm.i	Status table of VM _i in vStatusTable
cldStatusTable	List that stores set of cld and their status with SLA levels
cldStatusTable-j	Status table of cld _j in cldStatusTable
cldList	List of cld to be scheduled
cldSList	A map (cld, VM _i) to store scheduled cloudlets
cldTList	List to store temporary clds
maxVShare	VM with maximum computation share from all VMs
maxCld	Maximum length cld from cldList
vEFT	VM that gives earlier finish time for maxCld
tLength	Sum of clds length in cldList
isFinishedExecution	A flag to determine if a VM completes all assigned clds
slaLevel	SLA priority level of cloudlet (SLA-1, SLA-2, SLA-3)
slaWeight	Weight factor based on SLA level for VM share calculation
borrowedCapacity	Amount of capacity borrowed from other VMs
loadVariance	Variance in load distribution across VMs
rejectionRatio	Ratio of rejected cloudlets to total cloudlets
avgLoad	Average load across all VMs
balancePenalty	Penalty factor for load imbalance
dynamicCloudletArrived	Indicating arrival of new cloudlets during simulation
arrivalTime	Time when cloudlet arrives in the system
utilizationFactor	Factor based on current VM utilization
optimalVM	VM selected based on optimal scoring function

As demonstrated in Equation (3), the system calculates service-level adjustment coefficients for virtual machine resource distribution by utilizing the mean service priority values of currently assigned computational tasks.

$$\text{utilizationFactor}_j = \left(1.0 - \frac{\text{utilization}_j}{\text{totalUtilization}} \right) \times 2.5 \quad (4)$$

Equation (4) establishes a workload-dependent adjustment multiplier that favors lightly loaded virtual machines during workload redistribution operations.

$$\text{score}_j = (\text{utilization}_j \times 0.3) + (1.0 - \text{remainingCapacity}_j) \times 0.6 + \text{balancePenalty}_j \times 0.1 \quad (5)$$

Equation (5) produces a composite virtual machine ranking metric that incorporates weighted contributions from instantaneous resource consumption levels, available processing headroom and workload distribution penalty factors.

$$\text{remainingCapacity}_j = \frac{\text{VMShare}_j}{\text{VM}_j.\text{MIPS}} \quad (6)$$

Equation (6) quantifies the proportional share of virtual machine processing power that remains unallocated and accessible for new workload assignments.

$$\text{balancePenalty}_j = \frac{|\text{utilization}_j - \text{avgLoad}|}{\text{loadVariance} + \varepsilon} \quad (7)$$

Equation (7) introduces corrective weights for virtual machines exhibiting workload variance from mean system utilization thereby encouraging uniform resource allocation throughout the computational infrastructure.

$$\text{optimalVM} = \min_{\forall j \in \{1,2,3,\dots,m\}} (\text{score}_j) \quad (8)$$

Equation (8) identifies the virtual machine possessing the lowest composite ranking value as the most suitable destination for computational task placement among the accessible resource inventory.

$$\text{loadVariance} = \frac{1}{m} \times \sum_{j=1}^m (\text{utilization}_j - \text{avgLoad})^2 \quad (9)$$

Equation (9) evaluates the infrastructure-wide workload dispersion heterogeneity to identify conditions necessitating corrective redistribution actions.

$$\text{loadVariance} > \text{threshold} \quad (10)$$

The framework initiates preventive workload redistribution protocols when measured dispersion levels surpass established threshold boundaries signaling substantial resource allocation asymmetry.

$$\text{borrowedCapacity} = \sum_{j \neq \text{target}} \min(\text{VMShare}_j \times 0.95, \text{requiredLength} - \text{VMShare}_{\text{target}}) \quad (11)$$

As demonstrated in Equation (11), the system permits critical priority workloads to acquire up to 95% of idle processing capacity from alternate virtual machines during scenarios of inadequate local resource availability.

$$\text{SLA Cloudlet CT}_{i,j} = \frac{\text{Cloudlet}_i.\text{MI}}{\text{VM}_j.\text{MIPS}} + \text{VM CT}_j \quad (12)$$

Equation (12) determines the anticipated execution duration for computational tasks on designated virtual machines by incorporating workload magnitude and processing velocity parameters.

$$vEFT = \min_{j \in \{1,2,3,\dots,m\}} (SLA_Cloudlet_CT_{i,j} \times 0.3 + utilization_j \times 0.6 + balancePenalty_j \times 0.1) \quad (13)$$

Equation (13) identifies the virtual machine offering the most expeditious task finalization through a composite metric integrating execution duration projections, resource consumption levels and workload equilibrium adjustment factors.

$$executedMI_j = VM_j.MIPS \times updateTimeDifference \quad (14)$$

Equation (14) quantifies the aggregate instruction throughput in millions processed by virtual machines across the temporal window separating successive state monitoring cycles.

$$remainingMI_j = totalAssignedMI_j - executedMI_j \quad (15)$$

Equation (15) revises the outstanding computational burden on virtual machines by deducting completed instruction counts from the cumulative allocated task volume.

$$readTime_j = readTime_j - updateTimeDifference \quad (16)$$

Equation (16) recalibrates virtual machine availability timestamps by reducing them proportional to the duration elapsed since the preceding status synchronization ensuring precision in scheduling determinations.

$$rejectionRatio = \frac{Rejected_Cloudlets}{Total_Cloudlets} \quad (17)$$

Equation (17) quantifies system admission performance through the proportional metric of declined workloads resulting from insufficient computational resources or service-level agreement breaches.

$$ARUR = \frac{\left(\sum_{j=1}^m makeSpan_j / m \right)}{\max(makeSpan_j)} \quad (18)$$

Equation (18) assesses infrastructure utilization effectiveness by computing the quotient of mean execution completion time relative to peak completion duration.

$$Energy_j = \sum_{i=1}^n \left[((1 - cpuUtil_i) \times 100 + cpuUtil_i \times 150) \times \frac{finishTime_i - startTime_i}{1000} \right] \quad (19)$$

Equation (19) determines aggregate power expenditure through a linear consumption framework incorporating processor utilization metrics and operational time intervals.

$$F[i, j] = \begin{cases} 1, & \text{if Cloudlet}_i \text{ is assigned to VM}_j \text{ based on SLA priority} \\ 0, & \text{Otherwise} \end{cases} \quad (20)$$

A binary variable denoting the assignment status of cloudlets to virtual machines, governed by service-level agreement priority rules as presented in Equation (20).

3.3. Performance Model

The performance evaluation framework incorporates seven critical metrics that collectively capture the algorithm's operational characteristics. This comprehensive assessment enables rigorous comparison against conventional scheduling approaches and reveals the inherent compromises within multi-objective optimization under SLA constraints.

The metric of **ARUR** shown in Equation (21) that defines the usefulness of computational capacity of datacenter infrastructure is the following

$$\text{ARUR} = \frac{1}{n} \times \sum (U_i) \quad (21)$$

where U_i denotes the utilization rate of the i th resource and n represents the total resource count. **ARUR** values approaching 1.0 indicate balanced load distribution with proportional VM contributions while values near 0 signify load concentration on limited resources, resulting in underutilization of remaining capacity. **Throughput** denotes the total completed cloudlets over the simulation period given by Equation (22).

$$\text{Throughput} = \frac{\text{Number of Completed Cloudlets}}{\text{Total Simulation Time}} \quad (22)$$

High throughput indicates efficient resource coordination that reduces idle periods, optimizes task-to-VM matching and maximizes virtual machine utilization. **Makespan** is defined as

$$\text{Makespan} = \max(\text{FinishTime}_i) \quad (23)$$

Lower makespan values indicate superior scheduling efficiency through enhanced parallel processing and reduced delays. **Energy consumption** serves as a critical metric for evaluating the environmental and operational sustainability of scheduling systems, calculated using Equation (24)

$$E = \int P(u(t)) dt \quad (24)$$

where $P(u(t))$ represents the power consumption as a function of CPU utilization $u(t)$. The analysis employs a linear power model as expressed in Equation (25)

$$P(u) = P_{\text{idle}} + (P_{\text{max}} - P_{\text{idle}}) \times u \quad (25)$$

where P_{idle} represents the power consumption during idle state, P_{max} denotes the maximum power at full CPU utilization, and u ranges from 0 to 1.

The **Job Rejection Ratio** (JRR) is calculated as $\text{JRR} = (\text{Rejected Jobs} / \text{Total Submitted Jobs}) \times 100$. Lower JRR values signify effective workload management and indicate the system's ability to handle resource contention, maintain reliability and ensure SLA compliance.

Execution time is calculated by : $\text{Cloudlet Length} / \text{VM MIPS} = \text{Cloudlet Length} / \text{Million Instructions (MI)}$ $\text{VM MIPS} / \text{Million Instructions Per Second}$. The total execution time represents the cumulative processing duration for all completed cloudlets as expressed in Equation (26)

$$\text{Total Execution Time} = \sum_{i=1}^n \left(\frac{\text{Cloudlet Length}_i}{\text{VM MIPS}_i} \right) \quad (26)$$

Execution cost metric evaluates scheduling effectiveness through monetary quantification of cloudlet processing expenses across VMs. Cost calculation is formulated in Equation (27).

$$\text{Individual Cloudlet Cost} = \text{Cost Per Unit Time}_{\text{VM}} \times \left(\frac{\text{Cloudlet Length}}{\text{VM MIPS}} \right) \quad (27)$$

where VM_price represents the cost per unit time of the virtual machine and $\text{Cloudlet Length}/\text{VM MIPS}$ indicates the cloudlet's execution duration. The total execution cost aggregates the individual costs across all completed cloudlets as expressed in Equation (28)

$$\text{Total Execution Cost} = \sum \left(\text{Cost Per Unit Time}_{\text{VM}} \times \frac{\text{Cloudlet Length}}{\text{VM MIPS}} \right) \quad (28)$$

The cost model in Equations (27) and (28) intentionally computes execution cost as the product of a static VM cost-per-unit-time parameter and task execution duration. This formulation was adopted for some reasons: (i) it maintains methodological consistency with predecessor algorithms SLA-RALBA and DE-RALBA, ensuring fair cross-algorithm cost comparisons (ii) CloudSimPlus simulation framework does not natively model real-world cloud billing components such as data transfer fees, storage charges or dynamic pricing mechanisms making it methodologically appropriate to restrict cost evaluation to the computational execution dimension that the simulator can reliably model and (iii) it aligns with the broader CloudSim-based scheduling literature [9,10,16,23], providing a common baseline for cross-study comparison.

3.4. SLA-DE-RALBA Algorithm

SLA-DE-RALBA operates through a dual-scheduler architecture consisting of the Main Scheduler (presented in Algorithm 1) and the Update Scheduler (detailed in Algorithm 2). The simulation source code for SLA-DE-RALBA is publicly available online.⁴

Main Scheduler The SLA-DE-RALBA algorithm integrates SLA-aware task scheduling, dynamic resource allocation and load balancing across heterogeneous cloud environments. It processes cloudlet requests using two primary inputs: a cloudlet status table containing task requirements, SLA levels, timestamps and a VM status table with resource availability, utilization and capacity information.

Initialization: VM computational resources are quantified using MIPS metrics with dynamic share provisioning governed by task priority weights and real-time utilization patterns as presented in Equation(29).

$$\text{VMShare}_j = \text{ComputationRatio}_j \times \text{TotalMI} \times \text{SLAWeight}_j \times \text{UtilizationFactor}_j \quad (29)$$

Main Loop: The algorithm iterates until all cloudlets are either scheduled or rejected:

1. **Optimal VM Selection:** Determines the best VM candidate through capacity optimization, utilization balancing, and equitable load distribution.
2. **SLA-Aware Scheduling:** Selects cloudlets based on SLA priority (SLA Level 3 > 2 > 1).

⁴ https://bitbucket.org/sladeralba_26/dynamic_sla-de-ralba/src/main/

3. **Task Assignment:** Executes cloudlet-to-VM binding via `assignCloudletToVM` followed by synchronization of resource and cloudlet status tables.

Algorithm 1: SLA-DE-RALBA Main Scheduler

Input : List *cloudletStatusTables*, List *vmStatusTables*
Output: Optimally scheduled cloudlets with SLA compliance
Initialize simulation parameters and load dataset;
Calculate computation ratio and VM shares for all VMs;
while *cloudletList* $\neq \emptyset$ **do**
 optimalVM $\leftarrow$ *getOptimalVMForCloudlet*();
 if *optimalVM* $\neq$ null **then**
 targetCloudlet $\leftarrow$ *getMaxCloudletForVmShareWithSLA*(*optimalVM*);
 if *targetCloudlet* $\neq$ null **then**
 assignCloudletToVM(*optimalVM*, *targetCloudlet*);
 updateVMStatusTable(*optimalVM*);
 else if *dynamicCloudletArrived* **then**
 updateVMStatusTable(*currentTime*);
 dynamicCloudletArrived $\leftarrow$ false;
 else
 maxCloudlet $\leftarrow$ *getMaxCloudletWithSLA*() (SLA-1 > SLA-2 > SLA-3
 priority);
 if *maxCloudlet* $\neq$ null **then**
 targetVM $\leftarrow$ *getVMWithMinFinishTime*(*maxCloudlet*);
 if *targetVM*.*vmShare* < *maxCloudlet*.length **then**
 performLoadBalancing();
 borrowedCapacity $\leftarrow$ *borrowCapacity*(*targetVM*,
 maxCloudlet.length);
 assignCloudletToVM(*targetVM*, *maxCloudlet*);
 updateVMStatusTable(*targetVM*);
 else
 REJECTED_CLOUDLETS++;
 removeUnSchedulableCloudlet();
 else
 REJECTED_CLOUDLETS += *cloudletList*.size();
 break;
 procedure *onCloudletArrival*:
 if *newCloudletsArrived* **then**
 cloudletList.addAll(*newCloudlets*);
 dynamicCloudletArrived $\leftarrow$ true;
 goto step 3;
return *schedulingResults*;

Dynamic Adaptation: Processes incoming tasks by invoking *updateVMStatusTable*() to refresh resource states and queuing new cloudlets according to SLA priority enabling real-time workload adaptation.

Resource Management: If the optimal VM is unavailable the algorithm selects the highest-priority cloudlet using `getMaxCloudletWithSLA()` and identifies the VM with the earliest completion time via `getVmWithMinFinishTime()`. In capacity-constrained scenarios `performLoadBalancing()` redistributes tasks while `borrowCapacity()` leverages up to 95% of available capacity from other VMs to minimize rejections.

Rejection Handling: Identifies and removes unschedulable tasks via `removeUnschedulableCloudlet()`, increments the rejection counter and records rejection statistics for subsequent performance evaluation.

Event Handling: Monitors incoming cloudlet arrivals through `onCloudletArrival()`, enqueues new tasks and triggers priority queue reorganization to maintain SLA-based ordering.

Termination: Returns comprehensive performance metrics including scheduled cloudlet count, rejection ratio, VM utilization, execution times and resource consumption. These metrics facilitate evaluation across multiple dimensions such as ARUR and throughput demonstrating balanced achievement of cloud scheduling objectives.

VM Status Update The VM Status Update mechanism (illustrated in Algorithm 2)

Algorithm 2: VM Status Update

Input : `int currentTime`, `List vmStatusTables`

Output: Updated VM status tables

foreach `vmTable` in `vmStatusTables` **do**

`updateTimeDifference` $\leftarrow$ `currentTime` – `vmTable.lastUpdatedTime`;

if `vmTable.readTime` – `updateTimeDifference` ≤ 0 **then**

`vmTable.readTime` $\leftarrow 0$;

`vmTable.remainingMI` $\leftarrow 0$;

`vmTable.executedMI` $\leftarrow$ `vmTable.totalAssignedMI`;

else

`executedMI` $\leftarrow$ `vmTable.mips` $\times$ `updateTimeDifference`;

`vmTable.readTime` – `updateTimeDifference`;

`vmTable.executedMI` + `executedMI`;

`vmTable.remainingMI` $\leftarrow$ `vmTable.totalAssignedMI` – `vmTable.executedMI`;

`vmTable.lastUpdatedTime` $\leftarrow$ `currentTime`;

Recalculate VM shares with SLA weights;

`totalMI` $\leftarrow$ `calculateTotalMI()`;

foreach `vm` in `vmStatusTables` **do**

`slaWeight` $\leftarrow$ `calculateSlaWeight(vm)`;

`utilization` $\leftarrow$ `vm.remainingMI` / `vm.mips`;

`vmShare` $\leftarrow$ `vm.computationRatio` $\times$ `totalMI` $\times$ `slaWeight` $\times$ (1.0 – `utilization`);

`vm.vmShare` $\leftarrow$ `max(0, vmShare)`;

Load balancing check;

`loadVariance` $\leftarrow$ `calculateLoadVariance(vmStatusTables)`;

if `loadVariance` $\geq$ `threshold` **then**

`rebalanceVMs()`;

return `vmStatusTables`;

coordinates with the Main Scheduler to maintain real-time resource state accuracy ensuring scheduling decisions leverage current availability metrics rather than stale data. This enables responsive adaptation to evolving workload conditions including task terminations, new task submissions, and fluctuating VM resource consumption.

The update procedure accepts two inputs: the current simulation time and the comprehensive VM status table. It iteratively processes each VM entry (foreach vmTable in vmStatusTables) by computing the temporal delta: $\text{updateTimeDifference} = \text{currentTime} - \text{vmTable.lastUpdatedTime}$. Upon detecting task completion ($\text{vmTable.readTime} - \text{updateTimeDifference} \leq 0$), the algorithm resets the VM state with $\text{readTime} = 0$, $\text{remainingMI} = 0$, $\text{executedMI} = \text{totalAssignedMI}$, and aligns lastUpdatedTime to currentTime .

During each update cycle, the algorithm recalculates dynamic VM shares with SLA-weighted prioritization. The aggregate computational workload across the entire system is computed as:

$$\text{aggregateMI} = \text{calculateTotalMI}()$$

Individual VM workload contributions are then determined using Equations (30) and (31).

$$\text{slaWeight} = \text{calculateSlaWeight}(\text{vm}) \quad (30)$$

$$\text{SLAWeight}_j = 1.0 + \frac{\text{avgSLA}_{\text{VM}_j}}{10.0} \quad (31)$$

where $\text{avgSLA}_{\text{VM}_j}$ denotes the mean SLA priority level across all cloudlets allocated to VM j . The instantaneous utilization metric is computed according to Equation (32).

$$\text{utilization} = \frac{\text{vm.remainingMI}}{\text{vm.mips}} \quad (32)$$

Also the compute core share is calculated in Equation (33)

$$\text{vmShare} = \text{vm.computationRatio} \times \text{totalMI} \times \text{slaWeight} \times (1.0 - \text{utilization}) \quad (33)$$

A load-balancing term ($1.0 - \text{utilization}$) weights the distribution inversely with current utilization assigning higher shares to underloaded VMs and fostering equitable load distribution. Share values are bounded to prevent negative allocations through the constraint defined in Equation (34).

$$\text{vm.vmShare} = \max(0, \text{vmShare}) \quad (34)$$

Finally, the algorithm assesses aggregate load distribution across the system through Equations (35) and (36).

$$\text{loadVariance} = \text{computeLoadVariance}(\text{vmStatusTables}) \quad (35)$$

$$\text{LoadVariance} = \frac{1}{m} \times \sum (\text{utilization}_j - \text{avgLoad})^2 \quad (36)$$

When $\text{loadVariance} \geq \text{threshold}$ (nominally 0.02) which is an indicator of critical load imbalance, the algorithm calls `rebalanceVMs` to replace saturated VM loads with under-utilized resources without violating SLA constraints. New `vmStatusTables` ensure that future scheduling tasks are based on up-to-date information on resource availability. Such a continuous equilibration mechanism along with the Main Scheduler would allow SLA-DE-RALBA to offer adaptive features of effective resource utilization, load balance over time in the context of simulation run and stable SLA satisfaction in the case of heterogeneous workload.

4. Computing Setup

In this section, the cloud infrastructure was described as well as experimental datasets that will be used during the study.

4.1. Datasets

This study employs two established scientific benchmark datasets for experimentation: 1) **GoCJ dataset**, and 2) **HCSP dataset**. The GoCJ dataset is publicly available online on URL⁵, and HCSP Dataset is also available publicly online on URL⁶. The composition of datasets are also presented in our earlier studies in [9,16,23,30].

The original GoCJ benchmark dataset was introduced by Hussain and Aleem [32] and provides Google-derived realistic workload patterns representing real cloud job traces. To enable SLA-aware scheduling evaluation, the original dataset was modified as follows. The modified dataset retains the original job computational lengths expressed in Million Instructions (MI) ranging from lightweight operations of 100 MI to computationally intensive workloads of up to 10,000 MI. To facilitate hierarchical service differentiation and dynamic workload simulation, we structured the datasets in CSV format with three essential attributes:

1. **Cloudlet Length:** This metric quantifies the processing requirements of each task using Million Instructions (MI) as the unit of measurement. Our dataset includes tasks ranging from lightweight operations requiring 100 MI to computationally intensive workloads demanding up to 10,000 MI, enabling performance evaluation across diverse computational scenarios.
2. **SLA Level:** Implemented a three-level classification system to govern resource prioritization. Level 3 represents premium service with high-priority status and a 30% capacity allocation bonus. Level 2 offers a balanced approach, providing moderate priority with a 20% capacity enhancement. Level 1 caters to cost-optimized services, granting standard priority with a 10% capacity supplement. This tiered structure enables differentiated resource distribution based on service requirements.
3. **Arrival Time:** This parameter captures when individual tasks enter the simulation environment, allowing us to model dynamic workload behaviors and time-dependent scheduling constraints accurately.

Prior to the simulation, we applied the following preprocessing steps to the dataset: (i) rows containing non-positive job lengths or negative arrival times were identified and discarded (ii) SLA level values outside the valid range of 1–3 were flagged and removed (iii) arrival times were verified to be consistent with the simulation termination boundary of 5000 seconds and (iv) jobs were sorted by arrival time to ensure correct sequential injection during simulation. No imputation or synthetic augmentation was applied; only structurally valid records were retained for evaluation.

The **HCSP dataset**, which was originally proposed by Tracy [33] to use in our experiment. This was done by adaptation that included changing the data to test the SLA-DE-RALBA in our three-tier SLA system under different heterogeneity levels. The data

⁵ <https://data.mendeley.com/datasets/r93dfjvvv6/>

⁶ <https://www.fing.edu.uy/inco/grupos/cecal/hpc/HCSP/index.htm>

set is in CSV format having three major columns Task computational requirements (in terms of Million Instructions) designation of assigned SLA tier (between 1 and 3) and task arrival time to be applied in dynamic workload simulation. The dataset has a total of nine different workload types and each of them possesses distinct resources consumption patterns.

CPU-Dominant Workloads: The *u.c-hilo* category has tasks with heavy CPU usage requirement coupled with high I/O usage and low memory usage such that the capacity to allocate CPU is the main consideration in SLA. Contrary to this *u.c-lohi* tasks can be characterized by intensive CPU processing required along with large memory requirements but a low level of I/O activity which requires memory mindful resource allocation of SLA resources.

I/O-Centric Workloads: Tasks that are classified as *u.i-hilo* have both a high I/O intensity and large CPU demand with a low memory demand that demand I/O-intensive scheduling policies. The *u.i-lohi* type provides an I/O load and a high memory load and consumes less CPU resources, posing a memory-I/O optimization to SLA management. Meanwhile *u.i-lolo* tasks are pre dominated with I/O operations that require minimal CPU and memory.

Storage-Intensive Workloads: The *u.s-hilo* Work loads are workloads with a high storage requirement and intensive workload with little memory usage that puts a strong focus on the storage-prioritized resource allocation. On the other hand *u.s-lohi* tasks are both storage intensive and have high memory requirements with little CPU processing making scenarios of SLA optimization in storage, storage-memory.

4.2. Experimental Setup

For the experimentation, the CloudSim Plus simulator was used on a computing system equipped with an Intel Core i7-6600U processor operating at 2.80 GHz with turbo boost support and 8 GB DDR4 RAM. To emulate a cloud infrastructure environment, a simulated datacenter comprising 30 physical host machines was developed, supporting between 30 and 120 virtual machines according to the experimental requirements. The virtual machines were configured with varying processing capabilities measured in MIPS (Million Instructions Per Second), which served as the primary performance metric for executing workloads derived from the GoCJ and HCSP benchmark datasets.

The implementation architecture incorporated several key components, including a datacenter broker responsible for resource allocation, an SLA-aware scheduling mechanism, dynamic cloudlet management functionality, support for real-time task arrivals, and a heterogeneous virtualized infrastructure consisting of 30 hosts and 120 virtual machines with diverse computational capacities.

5. Experimental Results

This section provides a performance assessment of the proposed SLA-DE-RALBA algorithm. The evaluation of SLA-DE-RALBA and other schedulings namely DE-RALBA, DRALBA, SLA-RALBA, RALBA, PSSLB, PSSELB and Dynamic Max-Min is conducted using the performance metrics i.e., makespan and average resource utilization ratio (ARUR), energy consumption, execution time and cost, and throughput. Each experiment

is repeated five times and the average values obtained from these iterations are used for the analysis and comparison purposes. The experiments are performed using two benchmark workloads i.e., GoCJ and HCSP.

5.1. Performance Assessment Results - GoCJ Dataset

The experiments are performed using GoCJ benchmark dataset [32] and these results for each parameter are presented in this subsection.

ARUR evaluates resource utilization efficiency in cloud systems. As shown in Figure 3, SLA-DE-RALBA maintains consistently high ARUR (0.969–0.987) across all workloads (1000–6000 jobs) with zero job rejections.

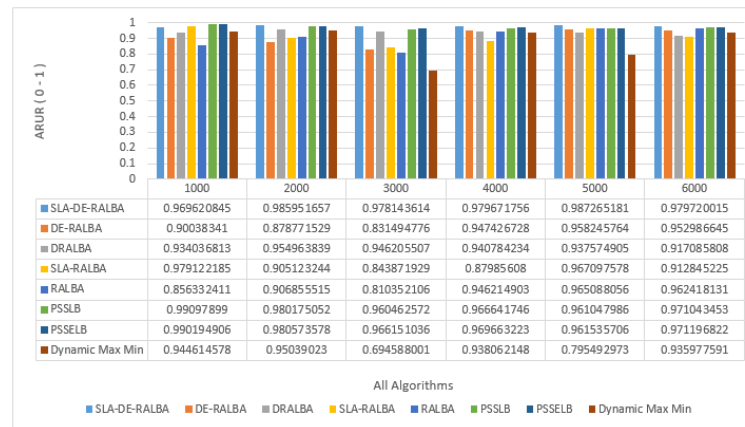


Fig. 3. ARUR using GoCJ dataset

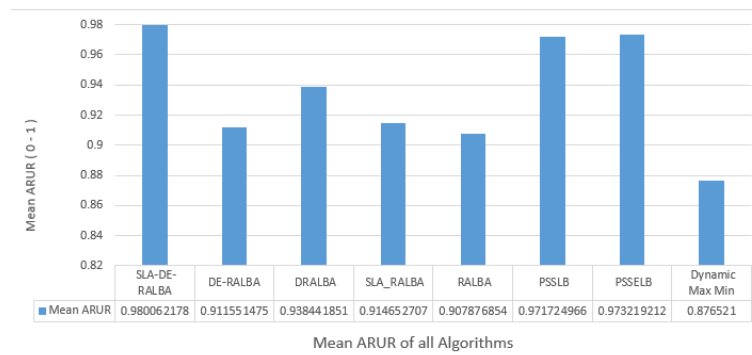


Fig. 4. Mean ARUR using GoCJ dataset

It slightly trails PSSLB and PSSELB at 1000 jobs but outperforms all baselines beyond 2000 jobs, with gains up to 28.9% over Dynamic Max-Min. At peak load (5000 jobs), it achieves the highest ARUR (0.987), confirming strong scalability and stable utilization. Overall, SLA-DE-RALBA delivers the best mean ARUR, improving over Dynamic Max-Min (11.9%), DE-RALBA (7.5%), RALBA (7.2%), SLA-RALBA (7.1%), DRALBA (4.5%), and marginally over PSSLB/PSSELB (0.8%). Figure 4 presents the mean ARUR results for all algorithms using GoCJ dataset.

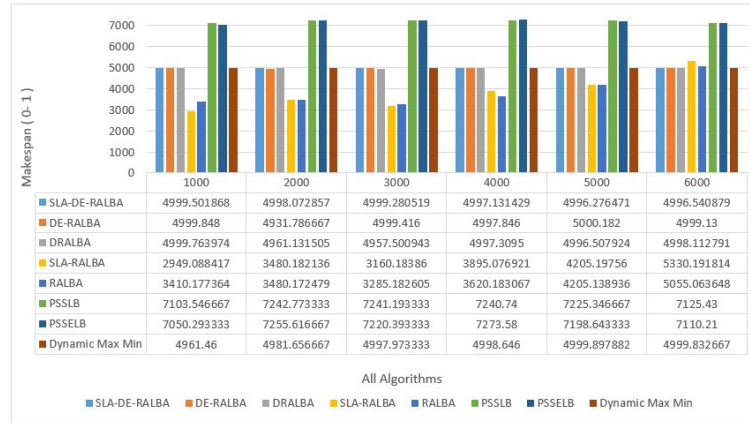


Fig. 5. Makespan using GoCJ dataset

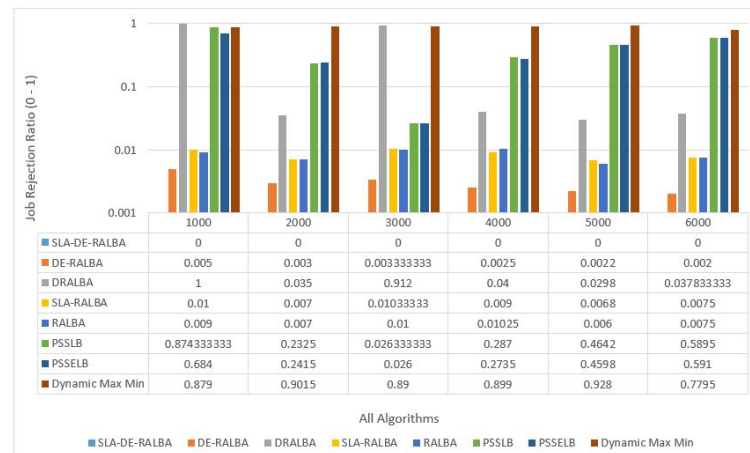


Fig. 6. Job Rejection Ratio using GoCJ dataset

Makespan measures overall job completion time, where lower values indicate better scheduling efficiency and faster execution, critical for meeting cloud service commitments.

As shown in Figure 5, SLA-DE-RALBA consistently achieves low makespan across all workloads. At 1000 jobs, it records 4999.50 time units, comparable to DE-RALBA and DRALBA, while for larger workloads (5000–6000 jobs), it remains stable between 4998–5000 time units. This stability is achieved through a multi-factor scheduling strategy that jointly considers resource availability, system utilization and SLA constraints at each decision point.

Figure 6 presents the Job Rejection Ratio (JRR), a key QoS metric measuring task rejections due to resource unavailability or deadline violations. SLA-DE-RALBA achieves zero job rejections across most GoCJ configurations, accepting more workloads than all baseline methods, while DRALBA shows the highest rejection rates at lower workload levels (1000 and 3000 jobs). This zero-rejection performance is enabled by three mechanisms: (1) capacity borrowing allowing high-priority tasks to use up to 95% of idle VM capacity, (2) proactive load rebalancing that redistributes tasks when VM load imbalance exceeds a threshold, and (3) dynamic VM share recalculation that adapts resources in real time with each incoming cloudlet.

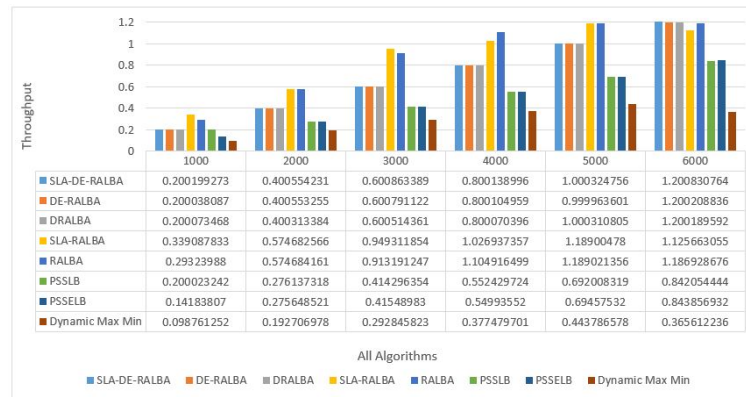


Fig. 7. Throughput using GoCJ dataset

Throughput measures the number of jobs completed over time, reflecting system processing capacity and overall efficiency. Higher throughput indicates better ability to handle concurrent workloads and improved service quality. As shown in Figure 7, SLA-DE-RALBA consistently achieves superior throughput across all workload scales, recording 0.20019 at 1000 jobs and maintaining strong performance at larger datasets. It significantly outperforms PSSLB and PSELB, enabling faster job completion and higher system efficiency. This improvement is driven by effective load balancing that maximizes task allocation and optimizes resource utilization.

Energy consumption reflects resource efficiency and sustainability in cloud computing. As shown in Figure 8, SLA-DE-RALBA maintains low and stable energy usage

(634–652 Joules) across all workloads (1000–6000 jobs). It achieves about 55–60% energy savings compared to PSSLB and PSSELB (1422–1435 Joules). Although Dynamic Max-Min consumes less energy (223–491 Joules), it does so at the cost of poor resource utilization, throughput, and execution performance.

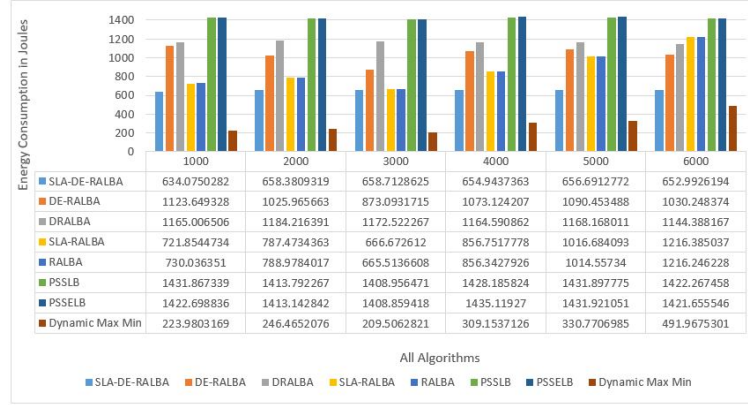


Fig. 8. Energy Consumption using GoCJ dataset

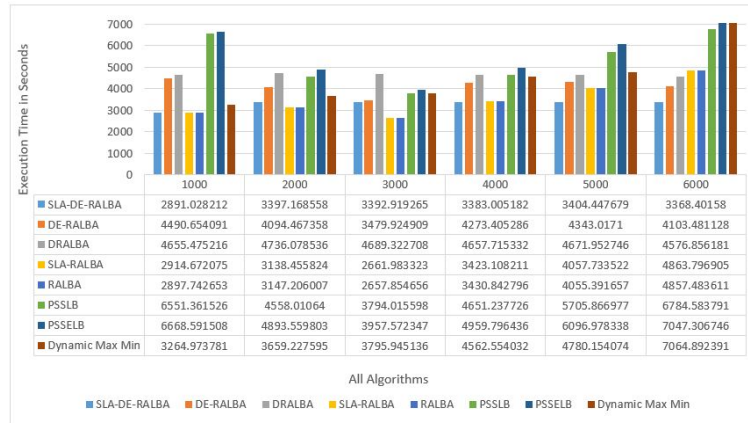


Fig. 9. Execution Time using GoCJ dataset

In Figure 9, SLA-DE-RALBA achieves consistently low execution time across all GoCJ workloads, with only minor variation (500 units), indicating strong scalability. This performance results from its SLA-based priority scheduling and multi-layer optimization, which prioritizes high-importance tasks while efficiently allocating resources.

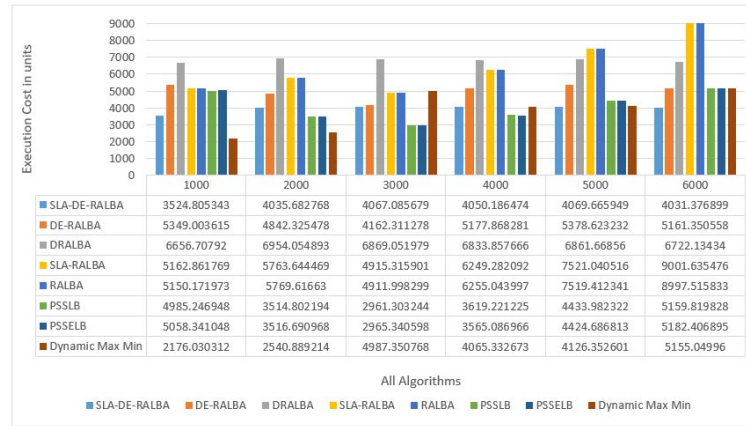


Fig. 10. Execution Cost using GoCJ dataset

The improved cost and resource utilization also reduce execution time, benefiting both users and cloud providers in terms of efficiency and cost savings.

Figure 10 shows that SLA-DE-RALBA is the most cost-efficient algorithm across all workload sizes, with costs ranging from 3524 units (1000 jobs) to 4100 units (6000 jobs), significantly lower than all baselines. Dynamic Max-Min is competitive only at low workloads but degrades at higher loads. PSSLB and SLA-RALBA show moderate cost efficiency, while RALBA, PSSELB, and DRALBA incur much higher costs, especially under heavy workloads, indicating poor scalability in cost optimization.

5.2. Performance Assessment Results - HCSP Dataset

The experiments are performed using HCSP benchmark dataset and these results for each parameter are presented in this subsection [33].

ARUR result in Figure 11 shows that SLA-DE-RALBA consistently achieves high resource utilization, with ARUR values exceeding 0.95 in centralized environments using HCSP dataset. RALBA and SLA-RALBA perform well under high-resource conditions, while PSSLB and PSSELB show strong results only when resources are abundant but degrade under constraints. DE-RALBA exhibits moderate performance in low-resource settings, whereas Dynamic Max-Min shows fluctuating results with inconsistent efficiency in distributed scenarios.

The makespan result (Figure 12) shows that SLA-DE-RALBA consistently achieves the best performance across different configurations, especially in U_CHIHI and U_S-LOLO scenarios. DE-RALBA maintains moderate and stable performance, while Dynamic Max-Min performs variably across conditions. PSSLB and PSSELB show moderate and consistent but less efficient makespan results.

Figure 13 shows that SLA-DE-RALBA maintains high job acceptance across all workload conditions, demonstrating strong reliability and effective workload handling. In contrast, DRALBA, DE-RALBA, SLA-RALBA, PSSLB, PSSELB, and Dynamic Max-Min exhibit varying levels of job rejection, especially under heavy loads, indicating limited

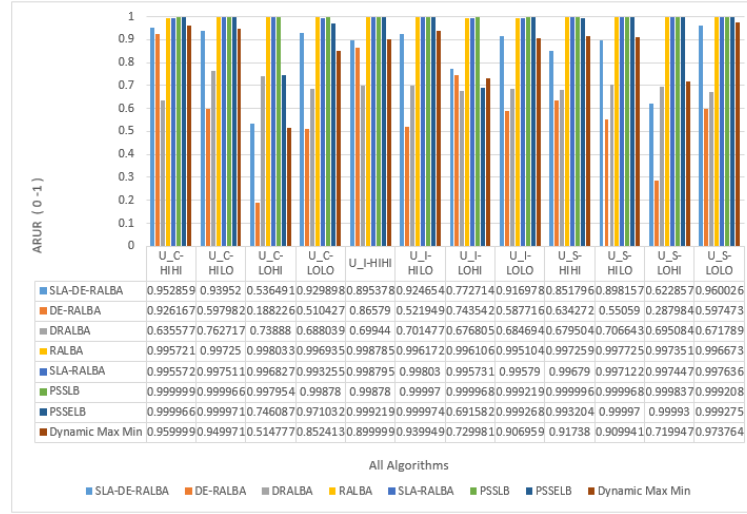


Fig. 11. ARUR using HCSP dataset

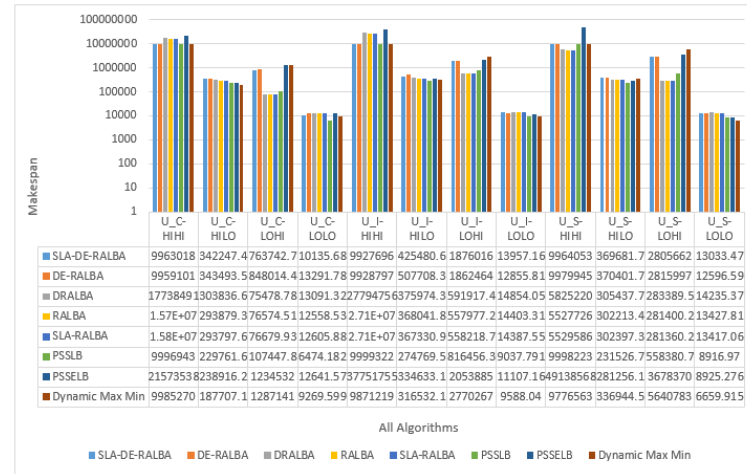


Fig. 12. Makespan using HCSP dataset

adaptability. Overall, SLA-DE-RALBA consistently ensures efficient job processing and SLA compliance across all configurations.

The throughput results using HCSP dataset (Figure 14) shows that SLA-DE-RALBA achieves optimal performance even under resource constraints. DE-RALBA, DRALBA, and RALBA maintain moderate, configuration-independent throughput, while PSSLB and PSSELB perform well only in LOLO scenarios. Overall, SLA-DE-RALBA delivers superior throughput, demonstrating the effectiveness of SLA-based scheduling for maximizing job completion efficiency. Energy consumption analysis (Figure 15) shows

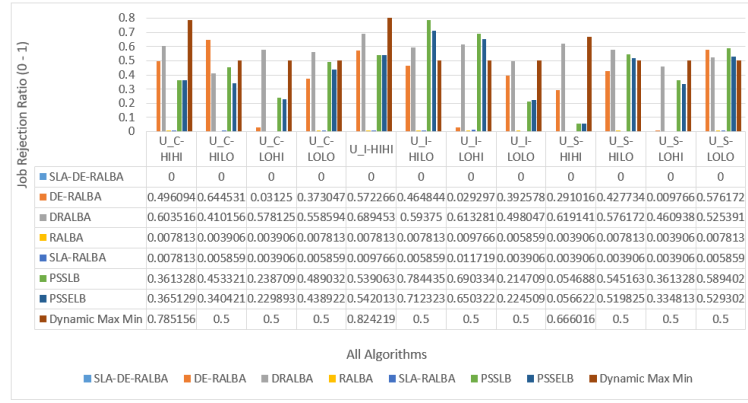


Fig. 13. Job Rejection Ratio using HCSP dataset

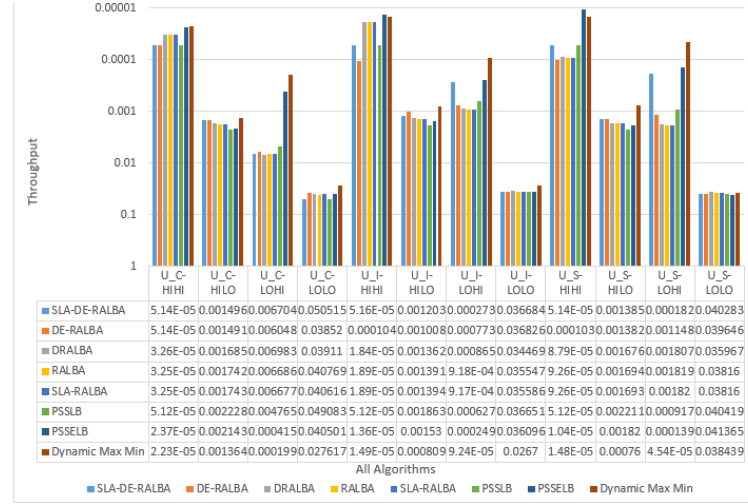


Fig. 14. Throughput using HCSP dataset

that RALBA and SLA-RALBA exhibit peak usage in U_IHIHI configurations. Dynamic Max-Min achieves the lowest energy consumption, reducing usage by 20–35% compared to RALBA/SLA-RALBA, 10–25% vs DE-RALBA, and 5–15% vs SLA-DE-RALBA, while PSSELB is the second most efficient with 15–30% savings. Although SLA-DE-RALBA is slightly less energy-efficient, it provides a better overall balance of resource utilization, makespan, and QoS than highly energy-optimized methods.

Figure 16 shows that SLA-DE-RALBA achieves consistently low execution times across both centralized and distributed models, setting the best benchmark performance. DE-RALBA provides stable moderate results, while PSSLB performs close to SLA-DE-RALBA. However, RALBA and SLA-RALBA exhibit significantly higher execution times in high-resource scenarios. PSSELB shows inconsistent performance, and DRALBA

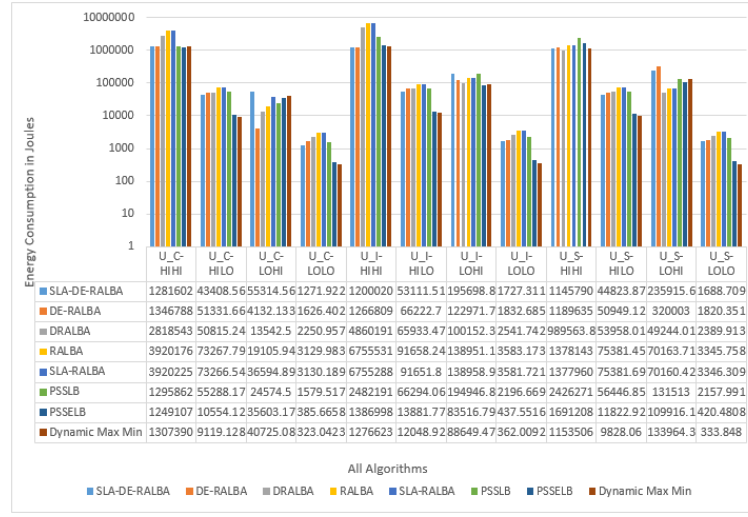


Fig. 15. Energy Consumption using HCSP dataset

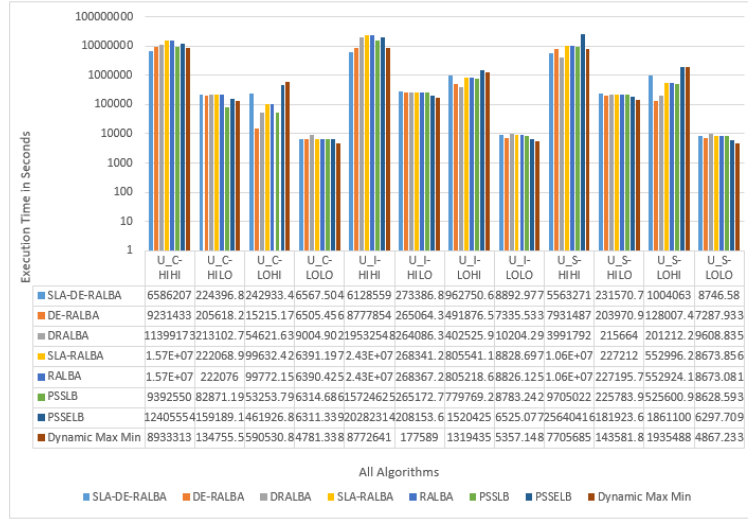


Fig. 16. Execution Time using HCSP dataset

experiences sharply increasing execution times in both centralized and scattered configurations.

Figure 17 shows that PSSLB achieves cost performance close to SLA-DE-RALBA in most cases. Dynamic Max-Min maintains generally low and stable execution costs, making it a cost-efficient option. In contrast, RALBA and PSSELB incur significantly higher costs in U_J-HIHI scenarios, while DRALBA and SLA-RALBA further increase costs under high-resource workloads.

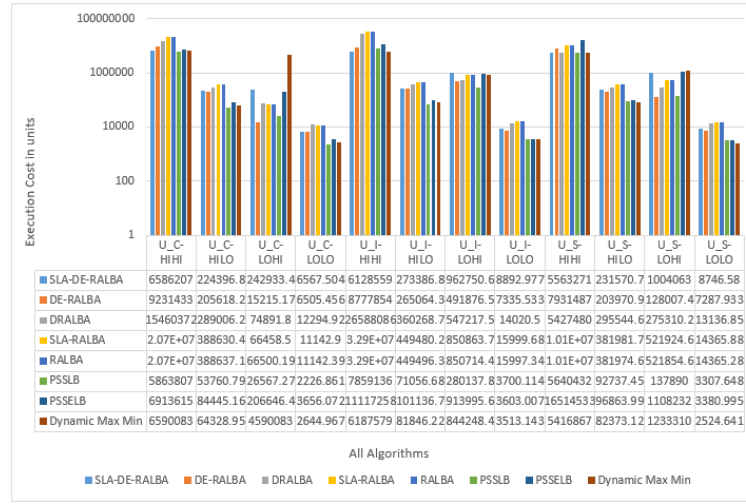


Fig. 17. Execution Cost using HCSP dataset

5.3. Results Discussion

SLA-DE-RALBA achieves a zero job rejection ratio across all GoCJ and HCSP workloads. In contrast, competing methods show varying rejection rates, ranging from minor levels in DE-RALBA to very high rejection in PSSLB, PSSELB, DRALBA, and Dynamic Max-Min. This consistent performance is due to its SLA-aware admission control and dynamic load balancing, which prevent resource saturation and ensure all jobs are successfully processed under diverse workload conditions.

SLA-DE-RALBA demonstrates consistent and robust performance across GoCJ and HCSP datasets by integrating multi-tier SLA scheduling with dynamic, multi-objective optimization for SLA compliance, cost, energy efficiency, and load balancing. **First**, its SLA-priority cloudlet selection ensures high-priority jobs are executed first, eliminating job rejections entirely ($JRR = 0$), unlike Dynamic Max-Min (50%), PSSLB (up to 87.4%), and DE-RALBA (up to 0.5%). This prevents resource saturation through explicit admission control. **Second**, a weighted VM scoring model balances execution time (25%), cost (35%), utilization (30%), and load balance (10%), enabling lower execution cost and time without sacrificing throughput or makespan, while avoiding server overloading or underutilization. **Third**, its Dynamic Enhanced adaptive tuning continuously adjusts scheduling decisions under changing workloads, keeping makespan stable (4996–4999) and outperforming PSSLB and PSSELB by 41%–45% in efficiency and 36%–57% in execution time. **Fourth**, reactive cloudlet migration during VM idle states improves real-time utilization, achieving a stable ARUR of 0.96–0.98 even under fluctuating or SLA-constrained workloads.

Finally, SLA-DE-RALBA achieves the highest mean ARUR (0.980) and lowest energy consumption (223–657 Joules), significantly outperforming PSSLB, PSSELB, DE-RALBA, and DRALBA in energy efficiency while maintaining superior utilization and

strict SLA compliance, making it the most balanced and reliable approach for real-world cloud environments.

6. Limitations

Although SLA-DE-RALBA demonstrates improved performance compared to existing scheduling algorithms, this study presents several limitations.

The evaluation is conducted exclusively in the CloudSimPlus simulation environment and has not been validated on physical testbeds, which may limit the representation of a few practical deployment factors such as network variability, hardware failures, system-level overheads, and multi-tenant resource contention. The algorithm does not explicitly incorporate hardware fault-tolerance or security mechanisms, as its design prioritizes energy efficiency and performance; consequently, its behavior under hardware failures or security-critical conditions remains unexamined.

Furthermore, the execution cost model, based on unit price and execution time, ensures methodological consistency and comparability but provides a simplified representation of real-world cloud pricing which may involve additional factors such as data transfer, storage, and dynamic pricing.

7. Conclusion

This study addresses key challenges in cloud job scheduling, including SLA compliance, workload variability, heterogeneous resources, and energy-cost trade-offs. Existing methods are either SLA-aware but static or dynamic but SLA-agnostic, limiting overall effectiveness.

To overcome this, we proposed SLA-DE-RALBA, implemented in the CloudSim Plus environment, integrating multi-tier SLA prioritization with dynamic optimization. Experimental results across multiple datasets show that SLA-DE-RALBA consistently outperforms existing algorithms, achieving the highest resource utilization ($ARUR = 0.980$), lower execution cost and makespan, and zero job rejection while maintaining stable SLA performance.

8. Future Work

There are a number of promising directions for the extension of this work. There are scalability studies that may be made to investigate the multi datacenter designs as well as the cloud fog edge paradigms of computation. Network and storage intelligence and computational resource could be part of the holistic resource optimization. The basic implementation would be tested in practice in platforms such as OpenStack or Kubernetes which would offer useful deployment information. Workload prediction and SLA- risk analysis could be improved by machine learning integration to increase forecast abilities. Multi- objective optimization models may facilitate detailed trade-offs between SLA compliance, energy-saving and cost-cutting. Lastly, it would be desirable to add fault tolerance features like failure identification and traffic redirection in order to enhance the reliability of the system in the most important applications.

References

1. Mell, P., Grance, T.: The NIST-National Institute of Standards and Technology- Definition of Cloud Computing. NIST Spec. Publ. 800-145, 1–3 (2011)
2. Armbrust, M. et al.: A view of cloud computing. *Commun. ACM* 53(4), 50–58 (2010). doi:10.1145/1721654.1721672
3. Marinos, A., Briscoe, G.: Community cloud computing. *Lect. Notes Comput. Sci.* 5931, 472–484 (2009). doi:10.1007/978-3-642-10665-1_43
4. Vaquero, L.M., Roderio-Merino, L., Caceres, J., Lindner, M.: A break in the clouds. *ACM SIGCOMM Comput. Commun. Rev.* 39(1), 50–55 (2008). doi:10.1145/1496091.1496100
5. Ajmera, K., Tewari, T.K., Singh, V.K., Vikash, Upadhyay, P.K.: Energy-Aware Dynamic Virtual Machine Scheduling in Cloud Computing: A Survey. In: *Proc. 2023 Fifteenth Int. Conf. Contemp. Comput. (IC3-2023)*, Noida, India, pp. 133–141 (2023). doi:10.1145/3607947.3607970
6. Awari, P.: Analyzing and Evaluating The Role of Cloud Computing in the Scalability of Web Applications. *Int. Res. J. Mod. Eng. Technol. Sci.* 7(2), 1–7 (2025)
7. Abraham, O.L., Bin Ngadi, M.A., Bin Mohamad Sharif, J., Sidik, M.K.M.: Task Scheduling in Cloud Environment-Techniques, Applications, and Tools: A Systematic Literature Review. *IEEE Access* 12, 138252–138279 (2024). doi:10.1109/ACCESS.2024.3466529
8. Koubàa, M., Karar, A.S., Bahloul, F.: Optimizing Scheduled Virtual Machine Requests Placement in Cloud Environments: A Tabu Search Approach. *Computers* 13(12), 1–25 (2024). doi:10.3390/computers13120321
9. Hussain, A., Aleem, M., Islam, M.A., Iqbal, M.: A rigorous evaluation of state-of-the-art scheduling algorithms for cloud computing. *IEEE Access* 6, 75033–75047 (2018). doi:10.1109/ACCESS.2018.2884480
10. Ibrahim, M. et al.: An In-Depth Empirical Investigation of State-of-the-Art Scheduling Approaches for Cloud Computing. *IEEE Access* 8, 128282–128294 (2020). doi:10.1109/ACCESS.2020.3007201
11. Rehan, H.: Revolutionizing America’s Cloud Computing the Pivotal Role of AI in Driving Innovation and Security. *J. Artif. Intell. Gen. Sci.* 2(1), 189–208 (2024). doi:10.60087/jaigs.v2i1.p208
12. Ebadifard, F., Babamir, S.M., Barani, S.: A Dynamic Task Scheduling Algorithm Improved by Load Balancing in Cloud Computing. In: *Proc. 2020 6th Int. Conf. Web Res. (ICWR)*, pp. 177–183 (2020). doi:10.1109/ICWR49608.2020.9122287
13. Albalawi, N.S.: Dynamic scheduling strategies for cloud-based load balancing in parallel and distributed systems. *J. Cloud Comput.* 14(1) (2025). doi:10.1186/s13677-025-00757-6
14. Chandrasiri, S., Meedeniya, D.: Energy-Efficient Dynamic Workflow Scheduling in Cloud Environments Using Deep Learning. *Sensors* 25(5) (2025). doi:10.3390/s25051428
15. Choudhary, A., Govil, M.C., Singh, G., Awasthi, L.K., Pilli, E.S.: Energy-aware scientific workflow scheduling in cloud environment. *Cluster Comput.* 25(6), 3845–3874 (2022). doi:10.1007/s10586-022-03613-3
16. Hussain, A., Aleem, M., Iqbal, M.A., Islam, M.A.: SLA-RALBA: cost-efficient and resource-aware load balancing algorithm for cloud computing. *J. Supercomput.* 75(10), 6777–6803 (2019). doi:10.1007/s11227-019-02916-4
17. Nabi, S., Ahmed, M.: OG-RADL: overall performance-based resource-aware dynamic load-balancer for deadline constrained Cloud tasks. *J. Supercomput.* 77(7) (2021). doi:10.1007/s11227-020-03544-z
18. Mostafa, S.A.M., Mohan, A., Wang, J.: SLA-MORL: SLA-Aware Multi-Objective Reinforcement Learning for HPC Resource Optimization. *arXiv:2508.03509* (2025). Available: <http://arxiv.org/abs/2508.03509>
19. Gogoi, B., Devaraj Verma C. D., Panggabean, C., Limbu, R., Sarker, R.: Optimized Cloud Resource Allocation Using Genetic Algorithms for Energy Efficiency and QoS Assurance. *J. Emerg. Technol. Innov. Res.* 12(5) (2025). doi:10.56975/jetir.v12i5.561700

20. Arshad, U., Aleem, M., Srivastava, G., Lin, J.C.W.: Utilizing power consumption and SLA violations using dynamic VM consolidation in cloud data centers. *Renew. Sustain. Energy Rev.* 167, 112782 (2022). doi:10.1016/j.rser.2022.112782
21. Mohammad, N.I.S.: Data Scheduling Algorithm for Scalable and Efficient IoT Sensing in Cloud Computing. arXiv:2508.04334 (2025). Available: <http://arxiv.org/abs/2508.04334>
22. Siddesha, K., Jayaramaiah, G.V.: Energy Efficient Greedy Scheduling of Tasks for DVFS Enabled Heterogeneous Multicore Processors. In: *Proc. 2021 6th Int. Conf. Recent Trends Electron. Information, Commun. Technol. (RTEICT)*, pp. 538–542 (2021). doi:10.1109/RTEICT52294.2021.9573873
23. Hussain, A., Aleem, M., Rehman, A.U., Arshad, U.: DE-RALBA: dynamic enhanced resource aware load balancing algorithm for cloud computing. *PeerJ Comput. Sci.* 11, 1–27 (2025). doi:10.7717/PEERJ-CS.2739
24. Alaei, N., Safi-Esfahani, F.: RePro-Active: a reactive–proactive scheduling method based on simulation in cloud computing. *J. Supercomput.* 74(2), 801–829 (2018). doi:10.1007/s11227-017-2161-0
25. Mao, Y., Chen, X., Li, X.: Max–Min task scheduling algorithm for load balance in cloud computing. *Adv. Intell. Syst. Comput.* 255, 457–465 (2014). doi:10.1007/978-81-322-1759-6_53
26. Nabi, S., Ibrahim, M., Jimenez, J.M.: DRALBA: Dynamic and Resource Aware Load Balanced Scheduling Approach for Cloud Computing. *IEEE Access* 9, 61283–61297 (2021). doi:10.1109/ACCESS.2021.3074145
27. Ma, Z., Ma, D., Lv, M., Liu, Y.: Virtual Machine Migration Techniques for Optimizing Energy Consumption in Cloud Data Centers. *IEEE Access* 11, 86739–86753 (2023). doi:10.1109/ACCESS.2023.3305268
28. Kumar, M., Sharma, S.C.: Dynamic load balancing algorithm to minimize the makespan time and utilize the resources effectively in cloud environment. *Int. J. Comput. Appl.* 42(1), 108–117 (2020). doi:10.1080/1206212X.2017.1404823
29. Chauhan, N. et al.: A Systematic Literature Review on Task Allocation and Performance Management Techniques in Cloud Data Center. *Comput. Syst. Sci. Eng.* 48(3), 571–608 (2024). doi:10.32604/csse.2024.042690
30. Hussain, A., Aleem, M., Khan, A., Iqbal, M.A., Islam, M.A.: RALBA: a computation-aware load balancing scheduler for cloud computing. *Cluster Comput.* 21(3), 1667–1680 (2018). doi:10.1007/s10586-018-2414-6
31. Shafiq, D.A., Jhanjhi, N.Z., Abdullah, A., Alzain, M.A.: A Load Balancing Algorithm for the Data Centres to Optimize Cloud Computing Applications. *IEEE Access* 9, 41731–41744 (2021). doi:10.1109/ACCESS.2021.3065308
32. Hussain, A., Aleem, M.: GoCJ: Google cloud jobs dataset for distributed and cloud computing infrastructures. *Data* 3(4), 1–12 (2018). doi:10.3390/data3040038
33. Tracy, D.B.: HCSP: heterogeneous computing scheduling problem (2000). Available: <https://www.fing.edu.uy/inco/grupos/cecal/hpc/HCSP/index.htm>
34. Arshed, J.U., Ahmed, M.: RACE: Resource Aware Cost-Efficient Scheduler for Cloud Fog Environment. *IEEE Access* 9, 65688–65701 (2021). doi:10.1109/ACCESS.2021.3068817
35. Venkatesh, C., Shiva Murthy, G.: Dynamic min-max algorithm for resource provisioning in cloud environment. *Indian J. Comput. Sci. Eng.* 12(3), 692–700 (2021). doi:10.21817/indjcse/2021/v12i3/211203161
36. Hagrass, T., El-Sayed, G.A.: A fault-tolerant and load-balancing scheduler for independent tasks on cloud-based virtual machines. *Cluster Comput.* 29(61), 1–13 (2026). doi:10.1007/s10586-025-05857-1
37. Zhu, X.: Hybrid ant colony and whale optimization algorithms for QoS-aware load balancing in cloud computing environment. *J. Eng. Appl. Sci.* 72(169), 1–26 (2025). doi:10.1186/s44147-025-00739-0

38. Beena, B.M., Ranga, P.C., Holimath, V., Sridhar, S., Kamble, S.S., Shendre, S.P., Priya, M.Y.: Adaptive energy optimization in cloud computing through containerization. *IEEE Access* 13, 159548–159565 (2025). doi:10.1109/ACCESS.2025.3606932

Mohsin Nawaz received the B.S. in Computer Science from the University of Engineering and Technology, Taxila, Pakistan and successfully completed his M.S. in Computer Science at the Department of Computer Science, KICSIT Campus, Institute of Space Technology, Islamabad, Pakistan. His research interests include software development and testing, cloud computing, and distributed computing, with a particular focus on performance analysis and resource optimization.

Altaf Hussain is serving as professor and Head of the Department of Computer Science at KICSIT, Institute of Space Technology (IST), Islamabad, Pakistan. He completed his PhD in Computer Science under the prestigious HEC Indigenous PhD Fellowship Program, with research contributions published in leading impact-factor journals including *IEEE Access*, *Cluster Computing*, *The Journal of Supercomputing*, *MDPI Data*, and *Computing and Informatics*. He has also contributed to international conferences and book chapters, demonstrating a strong and consistent research portfolio. He graduated with distinction in BS Computer Science and was awarded a Bronze Medal for academic excellence. He is reviewer for many journals and serves as an Academic Editor for *PLOS ONE*. His research interests include Information Security, Blockchain, Artificial Intelligence, Data Mining, Cloud Computing, High Performance Computing, Big Data Analytics, Software Engineering, and Outcome-Based Education, with a focus on scalable systems, performance optimization, and data-driven solutions for modern computing challenges.

Marran Al Qawaid received PhD in Computer Science from University of North Carolina. He is currently an Assistant Professor at Shaqra University, Saudi Arabia. His research interest includes cyber security, Artificial intelligence, Image processing, and computer vision.

Muazzam A. Khan Khattak is currently serving as a Member of the NADRA Authority Board and has been listed among the top 2% most highly cited scientists worldwide in 2024 and 2025. He is the recipient of Gold Medals from the Pakistan Academy of Sciences-COMSTECH and the IEEE Islamabad Section. At present, he is Director of the Farab Center for Artificial Intelligence and Cyber Security, Director of the ICESCO Chair in Data Analytics and Edge Computing and Director of Science Technology at Quaid-i-Azam University, Islamabad. He is also chairperson of UNESCO National IFAP Committee. Earlier, he served as Associate Dean of Computing and Head of the Postgraduate Program at SEECS, NUST. He completed his PhD through a sandwich program between IIUI and the University of Missouri, (UMKC), USA, followed by a postdoctoral fellowship at UMKC, where he later joined as an adjunct professor. Prof. Khattak has authored 3 books and 16 book chapters with leading international publishers, in addition to more than 240 articles in reputed international journals and conferences. His work has earned him recognition as one of the world's highly cited scientists, with over 7000 citations and an h-index of 45. His research focuses on Artificial Intelligence based smart solutions for Internet of Things, Internet of Vehicles, Smart Cities, and Information Security. He also holds adjunct

professorships at the University of Missouri (USA), Texas American Muslim Universities (USA), Harbin Institute of Technology (China), USAK University (Türkiye), BAKU State University (Azerbaijan), and Margullan University (Kazakhstan). He has been honored with the Pak Academy of Sciences Gold Medal, the Best Performance Award by the Ministry of Education in 2023, and multiple performance-based awards at NUST and QAU from 2015 till 2025, as well 2 best papers awards at International conferences. He has also been nominated for the prestigious civil awards Tamgha-e-Imtiaz and Sitara-e-Imtiaz. Over the course of his career, he has secured national and international research grants amounting to Rs. 1.7 billion from institutions including the Ministry of Education (Kazakhstan), the British Council, the University of Missouri (USA), USAK University (Türkiye), and Margullan University (Kazakhstan). Prof. Khattak serves on international advisory boards at TEXAM (USA), USAK University (Türkiye), Baku State University (Azerbaijan), Harbin Institute of Technology (China) and International Research Network on AI Development and Governance (AIR Net), China. He is also member Senate/BOG at Mirpur University of Science and Technology, AJK and Islamabad University of Health Sciences and Emerging Technologies, Islamabad.

Received: May 23, 2026; Accepted: June 23, 2026.

