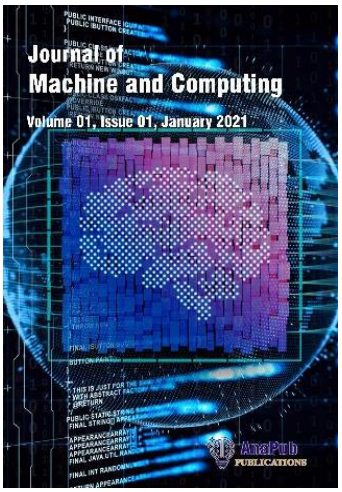
Journal Pre-proof

TSDACS: Two-Stage Deadline-Aware Cloudlet Scheduler for Time-Critical Workloads



Gritto D and Muthulakshmi P

DOI: 10.53759/7669/jmc202505161

Reference: JMC202505161

Journal: Journal of Machine and Computing.

Received 30 March 2025

Revised from 02 June 2025

Accepted 17 July 2025

Please cite this article as: Gritto D and Muthulakshmi P, "TSDACS: Two-Stage Deadline-Aware Cloudlet Scheduler for Time-Critical Workloads", Journal of Machine and Computing. (2025). Doi: https:// doi.org/10.53759/7669/jmc202505161.

TSDACS: Two-Stage Deadline-Aware Cloudlet Scheduler for Time-Critical Workloads

[1]D Gritto* , [2]P Muthulakshmi
[1,2] Department of Computer Science, Faculty of Science and Humanities,
SRM Institute of Science and Technology, Kattankulathur, Chengalpattu, Tamilnadu, India-603203.
Emails: grittodg@gmail.com, muthulap@srmist.edu.in
***Corresponding Author: D Gritto**

Abstract

In modern computing environments such as cloud, edge, and fog computing, as well as IoT networks and real-time systems, meeting workload deadlines is critical to ensure reliability, quality of service, and user satisfaction. The traditional scheduling algorithms often fail to adequately address the constraints associated with the workloads, particularly deadlines, the dynamic nature of workloads and resources, and the inherent resource limitations. Deadlines are the most important constraints, especially for time-sensitive applications. Achieving deadline compliance requires optimal resource provisioning, scheduling, resource allocation, resource scaling, workload migration, etc. This paper proposes a novel deadline-aware cloudlet scheduling algorithm, the Two-Stage Deadline-Aware Cloudlet Scheduling Algorithm (TSDACS), designed to minimize deadline misses through efficient resource provisioning and scaling strategies. In stage one, the algorithm provisions virtual machines with suitable configurations and quantities based on the requirements of the cloudlets to ensure they can be processed within their deadlines. Cloudlets are then scheduled onto the virtual machines in a way that minimizes deadline violations. In stage two, if the initially provisioned virtual machines fail to meet the deadlines, horizontal scaling is applied up to a limited threshold, to enhance the performance and the deadline compliance. Experimental results demonstrate that the proposed TSDACS algorithm outperforms existing approaches, such as CPDALB, DBS, and RDLBS2, in terms of deadline miss ratio, makespan, response time, and cost efficiency, while maintaining competitive VM utilization and effective load balancing.

Keywords: Resource Provisioning, Cloudlet Scheduling, Horizontal Scaling, Deadline-aware Scheduling.

1. Introduction

The growth of the internet and the widespread adoption of online business necessitated the development of advanced data processing and data storage technologies. Cloud computing addresses many of the limitations inherent in traditional monolithic computing systems confined to single machines or local networks. It enables users to leverage computing resources in an on-demand, scalable and cost-effective manner, providing seamless access to resources hosted in data centers operated by various cloud service providers (CSPs) [1]. Cloud platforms provision infrastructural units such as virtual machines (VMs), containers, and physical servers online. This provisioning allows users to autonomously utilize the resource components of these units, including RAM, CPU, operating system, bandwidth, and other network capabilities. Although cloud resources and services are pervasive, challenges related to workload execution and data storage on the cloud persist. The benefits of the cloud can be realized only when the services delivered by the CSPs align with user requirements, performance expectations, and Quality of Service (QoS) parameters. In practice, these three demands can be achieved through a combination of techniques, including resource provisioning, cloudlet scheduling, resource allocation, load balancing, auto-scaling, Service Level Agreement (SLA) management, VM migration, VM consolidation, fault tolerance and availability, data replication and backup, energy-efficient computing, performance monitoring and optimization, elastic resource management, and workflow management, among others. At the core of all these techniques lies task scheduling, or cloudlet scheduling. This paper uses tasks, cloudlets, and jobs interchangeably. Cloudlet scheduling is the fundamental process of mapping user workloads to the most appropriate provisioned resources or the VMs. By improving cloudlet scheduling, the effectiveness of all other

techniques is improved, leading to enhanced overall system performance, including better resource utilization, cost efficiency, and consistent QoS guarantees.

The scheduling algorithms are broadly classified into three basic categories: time-based schedulers, strategy-based schedulers, and objective-based schedulers. Time-based schedulers make scheduling decisions based on execution timing. Static schedulers determine the schedule before execution, whereas dynamic schedulers make decisions at runtime. Strategy-based schedulers focus on the approach used to derive a scheduling solution. Heuristic schedulers apply problem-specific rules or logic (e.g., Min-Min, Max-Min), while meta-heuristic schedulers employ general-purpose optimization strategies to explore the solution space (e.g., Genetic Algorithm, Particle Swarm Optimization, Ant Colony Optimization). Objective-based schedulers are designed to optimize specific goals, constraints, or quality measures. Examples include fault-tolerant scheduling algorithms, QoS-based scheduling algorithms, energy-efficient scheduling algorithms, and deadline constrained scheduling algorithms. The selection of a scheduling algorithm often depends on the specific requirements of the cloud application, the nature of the workloads, the constraints and the type of resources. In cloudlet scheduling, a constraint refers to any limitation, requirement, or condition that must be considered or satisfied when assigning cloudlets to available computational resources, such as VMs. Table 1 describes the various constraints that influence the cloudlet scheduling.

Deadline is a critical constraint in cloudlet scheduling as it ensures cloudlets are completed within an intended time frame, which is essential for meeting Service Level Agreements (SLAs) and maintaining system responsiveness, specifically in time-sensitive applications. The three main types of deadline-sensitive scheduling algorithms are hard deadline, soft deadline, and firm deadline algorithms. In hard deadline algorithms, cloudlets must be completed strictly before their deadlines, as any deadline miss can lead to system failure or unacceptable consequences. Soft deadline algorithms tolerate occasional deadline misses without critical impact. The system remains operational, although performance may degrade. In firm deadline algorithms, a cloudlet holds no value if completed after its deadline. While a deadline miss does not cause system failure, firm deadline algorithms discard the cloudlet, as its late execution is considered ineffective. The selection of a deadline-sensitive scheduling algorithm can also be based on the periodicity of the cloudlets. The real-time cloudlets that are executed or activated at regular, fixed time intervals are called periodic cloudlets. Each activation is referred to as a job, and these jobs repeat indefinitely. Non-periodic cloudlets are those that activate at irregular, unpredictable intervals. In multi-core real-time systems, ensuring energy-efficient execution of both periodic and aperiodic cloudlets with precedence constraints under energy harvesting constraints is briefed in [2]. The Maximum Miss First (MMF) algorithm dynamically prioritizes periodic tasks based on their historical deadline miss ratios to ensure fair QoS attainment in soft real-time systems. Hard Deadline Co-Evolutionary Genetic Algorithm (HDCGA) schedules workflow applications with strict deadlines in heterogeneous environments [3, 4]. Scheduling algorithms like Earliest Deadline First (EDF), Rate Monotonic Scheduling (RMS), and Deadline Monotonic Scheduling (DMS) work well for periodic cloudlets, while non-periodic cloudlets can be managed effectively with algorithms like EDF and Least Laxity First (LLF). EDF is suitable for both types of cloudlets.

EDF always selects the cloudlet with the nearest or earliest deadline for execution. LLF schedules the cloudlet with the least laxity (slack time) or the one closest to missing its deadline, where laxity = deadline − (current time remaining execution time). RMS grants higher priority to the cloudlets with shorter times and schedules them first. The time period of a periodic task refers to the fixed time interval between consecutive activations. The DMS algorithm works on relative deadlines. A relative deadline is the difference between the absolute cloudlet deadline and the activation time of the periodic cloudlet. Activation time is the time at which a cloudlet becomes available for execution. DMS assigns higher priority to the cloudlets with shorter relative deadlines and schedules them first. Modified versions of EDF, such as the Earliest Feasible Deadline First (EFDF) approach, reduce time complexity and the number of cloudlet migrations by using FIFO queues, processor affinity, and feasibility checks. The Delayed Rate-Monotonic (DRM) algorithm improves processor utilization and reduces cloudlet pre-emptions in real-time systems, demonstrating its superiority over the traditional Rate-Monotonic (RM) algorithm [5, 6]. The Improved Least-Laxity-First (ILLF) scheduling algorithm reduces cloudlet switching overhead by dynamically adjusting execution time slices for periodic cloudlets, proving more efficient than traditional LLF in minimizing pre-emptions [7]. The comparative analysis of the four algorithms reveals that EDF is optimal for balanced systems but performs poorly during overloads. LLF is also optimal but impractical due to excessive context

switches caused by frequent laxity updates. While RMS is simple, DMS improves upon it by supporting deadlines shorter than periods. RMS prioritizes tasks with the shortest time periods, whereas DMS prioritizes tasks with the shortest relative deadlines [8].

Table 1: Constraints Affecting Cloudlet Scheduling

Type of Constraint	Description	Examples
Resource constraints	Limitations related to the availability and consumption of resources.	Cloudlet resource requirements, VM capacity, cloud service provider resource limits, etc.
Cloudlet constraints	Requirements and restrictions specific to individual cloudlets.	Deadline, priority, cloudlet dependencies, QoS requirements, security requirements, etc.
VM constraints	Limitations or requirements related to virtual machines.	Isolation, compatibility, cost, performance guarantees, geographic location, etc.
Scheduling constraints	Constraints related to the algorithms and policies used for scheduling cloudlets to VMs.	Scheduling algorithms, load balancing policies, etc.
Cloud service provider constraints	Restrictions imposed by the cloud service provider.	SLAs, pricing models, policies and regulations, etc.
Additional constraints	Other factors that may influence scheduling decisions.	Energy efficiency, fault tolerance, scalability, etc.

Resource limitation poses a critical challenge in ensuring deadline compliance. To address this, several techniques, such as resource reservation, task splitting, task replication, dynamic resource scaling, and task migration among VMs, are used. The reservation-based technique involves pre-allocating specific computing resources or VMs exclusively for deadline-sensitive tasks. This advance reservation reduces resource contention and ensures task completion within its deadline. The task-splitting method divides complex tasks into smaller sub-tasks to ensure assignments meet their deadlines by enabling parallel execution. The replication technique runs multiple copies of the same task simultaneously on different VMs and uses the earliest completed result to enhance timeliness. Dynamic VM scaling allocates or deallocates VMs in real time driven by workload needs. Scaling plays a significant role in providing additional resources when tasks with tight deadlines are scheduled. Task migration helps meet deadline compliance by moving tasks from overloaded to underutilized VMs, reducing execution delays, and balancing resource usage. Greedy Reclamation of Unused Bandwidth-Power-Aware (GRUB-PA) is a power-aware scheduling algorithm based on resource reservation that dynamically adjusts processor voltage and frequency to reduce energy consumption while ensuring deadlines [9]. The Task Duplication-based Scheduling Algorithm (TDSA) proactively duplicates critical tasks to optimize performance without violating budget constraints [10]. The node scaling model for power-aware scheduling demonstrates that adjusting the number of cores and their speeds can minimize power consumption while meeting deadline constraints [11]. The migration-aware scheduling technique for multiprocessor systems prioritizes non-periodic tasks by migrating them to other processors if their deadlines permit, reducing both response time and energy consumption [12]. Several studies have explored dynamic VM scaling to optimize application performance and resource utilization. The AppRM tool automatically configures resource controls for both VMs and resource pools to meet application Service Level Objectives (SLOs) using reservations, limits, and shares techniques. ICLB compares vertical and horizontal scaling strategies in inter-cloud environments, demonstrating effective resource optimization through real-time workload monitoring and load balancing [13, 14]. VM migration performance has been enhanced through various approaches. A fuzzy inference-based framework analyzes factors such as dirty page rate and latency to reduce migration time and downtime. A distance-based traffic-aware algorithm improves scalability by minimizing round-trip time (RTT) through client proximity and low network traffic [15, 16]. Resource scaling and task migration are widely adopted and key techniques for ensuring deadline compliance. Task migration is typically costlier than scaling for several reasons. It involves moving entire tasks between VMs or physical hosts, including the transfer of memory, CPU state, and I/O buffers over the network, which introduces latency, bandwidth overhead, and potential service disruption. Both types of scaling generally outperform task migration. Horizontal scaling adds or removes VM instances, avoiding costly live state transfers, while vertical scaling adjusts CPU or cores within the same VM or host, making it more efficient than task migration by eliminating data transfer.

The proposed TSDACS algorithm employs optimal resource provisioning and controlled VM scaling to ensure deadline compliance in deadline-sensitive environments. TSDACS is a deadline-aware scheduling algorithm designed to optimize key performance metrics, such as deadline compliance, makespan, and cost. It also aims to enhance other important indicators, including response time, VM utilization ratio, and load balancing. The main contributions of this study include:

1. **Deadline-aware initial VM provisioning:** VMs are computed and provisioned based on cloudlet characteristics and deadline constraints, avoiding random or arbitrary configurations.
2. **Threshold-based horizontal scaling:** VM scaling is limited by a fixed threshold to control resource consumption, reduce costs, and prevent resource exhaustion for other users.
3. **Adaptive soft deadline scheduling:** When the scaling threshold is reached, the scheduler adapts by relaxing deadline constraints to avoid cloudlet failures or rejections.
4. **Dynamic feedback-driven scheduling:** The system continuously monitors cloudlet deadlines and dynamically adjusts scheduling and scaling decisions in response to workload variations.
5. **Efficient resource-constrained scheduling:** TSDACS maximizes deadline compliance even in environments with limited resources while optimizing additional performance measures such as makespan, cost, etc.

The remainder of this research paper is organized into six sections. Section 2 presents a literature review of contemporary deadline-sensitive cloudlet scheduling algorithms. Section 3 presents the problem model and the proposed framework. Sections 4 and 5 elaborate on the proposed methodology and the experimental setup, respectively. The results and discussion are presented in Section 6. Finally, Section 7 concludes the paper with final remarks and outlines potential directions for future enhancements.

2. Related Work

Scheduling cloudlets is a complex task that involves balancing various, often conflicting, performance, Quality of Service (QoS), and efficiency metrics. The schedulers must exhibit a balance between measures like makespan, turnaround time, response time, resource utilization, fault tolerance, energy efficiency, scalability, reliability, cost, etc. However, achieving an optimal balance between these measures is challenging due to inherent trade-offs. For instance, maximizing resource utilization can lead to increased energy consumption, while improving fault tolerance may reduce overall system efficiency. Therefore, selecting an effective scheduling algorithm often involves making compromises and selecting a strategy based on specific workloads, user requirements, or system constraints. In particular, scheduling algorithms are essential for meeting QoS requirements such as deadlines for ensuring timely cloudlet execution in deadline-sensitive environments. This review explores literature on scheduling approaches for deadline-sensitive cloudlets and evaluates their ability to meet deadline constraints.

2.1. Scheduling Deadline-Sensitive Cloudlets

Ebrollahi et al. modified the EDF algorithm by reserving extra time during scheduling. If any cloudlet fails due to a transient fault, it can be re-executed within the reserved time without missing its deadline. The Group Priority Earliest Deadline First (GPEDF) scheduling algorithm proposed by Qi Li et al. groups cloudlets with similar deadlines to reduce the number of priority levels. This approach improves scheduling efficiency, response time, and context-switching performance compared to traditional EDF [17, 18]. This paper introduces Min-Min II, an enhanced cloudlet scheduler based on the classic Min-Min algorithm that incorporates deadline awareness and communication delays to minimize makespan and deadline misses while enhancing VM utilization. Min-Min II immediately schedules cloudlets that can meet their deadlines on optimal VMs while deferring cloudlets that are violating deadlines by placing them in a waiting queue for later allocation. This methodology by Li-Ya Tseng et al. shows improved makespan and better deadline compliance

in contrast to the Min-Min algorithm. However, the performance of Min-Min II relies on execution time estimates, which may be inaccurate in real-world dynamic environments [19]. This paper presents a deadline-constrained workflow scheduling algorithm that optimizes cost and performance using Particle Swarm Optimization (PSO). Targeting scientific workflows like Montage and LIGO, Maria A. et al. dynamically provision heterogeneous VMs to handle varying workloads, pay-per-use billing, and task dependencies. By treating schedules as swarm particles and penalizing deadline violations, the method efficiently minimizes execution costs. However, the high computational overhead of PSO limits its scalability for large-scale workflows [20]. The Cost Deadline Based (CDB) algorithm by Himani et al. aims to reduce deadline misses and costs for both cloud users and providers. It uses an Earliest Deadline First (EDF) approach with Min-Min scheduling and space-shared policy, prioritizing cloudlets by deadlines and user payment limits. Net profit is estimated based on cloudlet parameters and VM costs. The algorithm improves profit and throughput and reduces losses compared to traditional methods. The flexibility of CDB is limited in dynamic environments with unpredictable execution times [21].

Suvendu Chandan Nayak et al. proposed a modified backfilling algorithm using the Vlsekovic criteria compromise (VIKOR) multi-criteria decision-making method to schedule deadline-based cloudlets. Cloudlets are ranked based on execution time and deadlines, with utility, regret, and compromise measures. The algorithm ensures optimal resource use and minimal deadline misses by scheduling cloudlets in ascending order of the compromise measure. However, the performance relies on accurate execution time and deadline estimates, which may limit its effectiveness in dynamic environments [22]. The energy-aware task scheduling with deadline constraints (EATSD) approach proposed by Ben Alla et al. is based on differential evolution and ELECTRE IV multiple-criteria decision-making methods, forming the DEEL model for dynamic cloudlet prioritization and VM allocation based on Fuzzy Logic and Particle Swarm Optimization techniques. Both the cloudlets and the VMs are fixed with priority based on task length, deadline, waiting time, burst time, and the speed of the VMs. EATSD optimizes energy consumption, reduces makespan, and ensures deadline adherence. VM migrations have the potential to influence energy savings and scheduling efficiency in dynamic cloud environments [23]. The Deadline Constraint-based Scheduling Algorithm (DCSA) introduced by Jianpeng Li et al. dynamically classifies cloudlets into regular, emergent, or invalid based on their remaining time before deadlines and estimated execution times. It assigns regular cloudlets to idle nodes, pre-empting cloudlets for urgent workloads while ensuring suspended cloudlets still meet deadlines, and discarding impossible-to-complete cloudlets. Theoretical analysis proves DCSA avoids thrashing and deadline misses due to excessive pre-emption. The algorithm balances urgency but treats all cloudlets as having similar resource requirements, ignoring potential variations in CPU, memory, or IO needs that could affect real-time scheduling decisions [24].

Samra Sahar et al. proposed the Learning Automata-based Scheduling (LAS) algorithm to minimize energy consumption and makespan for deadline-sensitive cloudlets. It uses adaptive learning automata to dynamically map cloudlets to VMs based on deadlines and VM heterogeneity, improving resource utilization and deadline compliance. Though effective, LAS may face scalability issues with large cloudlet sets and may result in reduced performance under highly dynamic workloads [25]. Anurina Tarafdar et al. present two cloudlet scheduling algorithms: Energy Makespan Aware (EMA), a greedy method minimizing Energy Makespan Cost (EMC) for energy-performance balance, and ACOEM, which enhances EMA via Ant Colony Optimization for better performance. Both employ three-tier (host-VM-cloudlet) optimization and dynamic scaling to meet deadlines efficiently. EMA offers quick decisions, while ACOEM delivers superior results through bio-inspired search. The proposed approach has two key limitations. First, cloudlet deadlines are artificially determined rather than derived from actual application requirements. Second, the methodology assumes all VMs of the same type have identical configurations [26]. The paper proposed by Yu Zhang et al. presents a deadline-aware dynamic scheduling method for edge-cloud systems in Industrial IoT, combining two algorithms: Dynamic Time-Sensitive Scheduling algorithm (DSOTS), which prioritizes cloudlets based on resource capabilities and deadlines, and the Time-Sensitive Greedy Scheduling algorithm

(TSGS), which improves latency and cost through intelligent load balancing. The approach achieves faster processing, lower costs, and fewer deadline violations than traditional schedulers, though greedy optimization and predictable cloudlet arrivals may result in suboptimal performance [27]. Xiaojian He et al. combined Enhanced Ant Colony Optimization (EACO) with Modified Backfilling (MBF) to efficiently schedule deadline-constrained cloudlets, balancing energy, makespan, and other QoS measures. The EACO scheduler assigns cloudlets to suitable VMs to optimize energy consumption and makespan while adhering to deadlines. MBF reorders cloudlets in VM waiting queues to improve the cloudlet completion rate. However, the method assumes static workloads and VM configurations, and the performance depends on the tuning parameters [28]. Table 2 provides the comparative study of the deadline-based scheduling algorithms available in the literature.

Table 2: Comparative Evaluation of Deadline-Based Scheduling Algorithms

Ref. No.	Algorithm/Technique	Performance Evaluated	Advantages/Features	Limitations Identified
[29]	Deadline-aware and Cost-effective Hybrid Genetic Task Scheduling (DCHG-TS): Genetic and Heterogeneous Earlier Finishing Time (HEFT) Algorithm.	Makespan, cost, load balancing, and deadline compliance.	Fast convergence, multi-objective optimization, and dynamic load balancing.	High computation and static deadlines.
[30]	Priority-aware Semi-Greedy (PSG) and PSG with Multi-start (PSG-M).	Deadline satisfaction percentage, energy consumption, deadline violation time, and makespan.	Mixed Integer Linear Programming (MILP) model, semi-greedy approach, and priority-aware cloudlet sorting results in energy optimization and minimize violations.	Static deadlines and single cloudlet execution per fog node.
[31]	Heuristic-Based Genetic Algorithms (HGAs): Bottom-level GA (BGA), Top-level GA (TGA), and Bottom-Top-level GA (BTGA)	Normalized schedule cost, deadline compliance, and execution time minimization	Priority-based initialization uses b-level (critical path) and t-level (earliest start time) for cloudlet prioritization. BTGA integrates both b-level and t-level for better diversity.	Static deadlines, high complexity, and limited scalability.
[32]	Adaptive Deadline-based Dependent Job Scheduling (A2DS)	Makespan, processor utilization, and starvation avoidance.	Two-tier VMs (foreground/background) optimize resources and minimize makespan. Resolves task dependencies, prioritizes deadlines, prevents deadlock, and improves resource utilization.	Deployment overhead, VM switching latency, and scalability.
[33]	Efficient Deadline and Priority Job Scheduling (EDPS).	Deadline compliance, execution time, resource utilization, and energy consumption.	Linear Programming Problem (LPP) for CPU selection and applying Shortest Execution First Scheduling (SEFS) for unconstrained jobs achieve a high deadline-meeting ratio and reduce VM usage.	Scalability and energy optimization trade-offs.
[34]	Hybrid cloud-based Mixed-Integer Linear	Cost minimization, deadline, and security compliance.	Formulated MILP workflow scheduling minimizes execution	Static workflow parameters and high computational

	Programming (MILP) model.		time, data transfer time, and costs while meeting deadlines and security constraints, and reduces inter-cloud communications for dependent cloudlets.	complexity for large workflows.
[35]	Fuzzy Priority Deadline (FPD) approach.	Deadline compliance, cost, makespan, degree of imbalance, and SLA violation count.	Combines fuzzy logic and heuristic-based cloudlet scheduling by dynamically determining and allocating the optimal number of VMs based on cloudlet characteristics, guaranteeing SLA, and ensuring deadline compliance and minimal cost.	Fuzzy controller tuning, scalability, cloudlet length specificity, and limited priority levels.
[36]	Adaptive Deadline-Based Scheduling (ADBS)	Deadline compliance, CPU utilization, turnaround time, and waiting time.	The dynamic priority assignment, or adjusting cloudlet priorities based on deadlines, deadline-aware time slicing, load balancing, pre-emption, and scalability improves the performance of the algorithm.	Computational overhead, task-length specificity, limited priority levels, and dependency on deadline accuracy.
[37]	Deadline and Cost-aware Genetic Algorithm (DCGA)	Success rate of meeting deadlines, execution time, and execution cost.	Considering cloud characteristics, a novel encoding method and improved population initialization, crossover, and mutation achieve high success rates and cost efficiency under deadlines.	Does not address VM failures, cloudlet reassignment, and the assumption of free data transfer between VMs.
[38]	Dynamic Scheduling of Bag of Tasks-based workflows (DSB).	Success rate of meeting deadlines and execution cost.	Grouping tasks into Bags of Tasks (BoTs) based on dependencies and priorities, using Mixed Integer Programming (MIP) for dynamic VM provisioning, and considering features like elasticity, heterogeneity, and VM provisioning delays achieve high success rates and cost efficiency within deadlines.	Reliance on perfect runtime estimates, IBM ILOG CPLEX, scalability, and single-objective optimization.

B. Comparison Algorithms for Performance Evaluation

The performance of the proposed TSDACS is compared against three deadline-aware algorithms: Capacity Based Deadline Aware Dynamic Load Balancing (CPDALB) algorithm, the Deadline Budget Scheduling (DBS) algorithm, and the Receiver Initiated Deadline Aware Load Balancing Strategy 2 (RDLBS2) algorithm proposed by Raza Abbas Haidri et al., Mokhtar A. Alworafi et al., and Raza A. Haidri et al., respectively.

In the CPDALB algorithm, the initial schedule is generated using the Min-Min scheduling algorithm. Each cloudlet is dynamically evaluated against two key conditions on its assigned VM: (1) whether

adding the cloudlet will not exceed the current VM load capacity and (2) whether the cloudlet can complete within its deadline. If both conditions are satisfied, the cloudlet remains on the current VM. If either condition fails, the system searches for an alternative VM that meets both requirements. When no suitable VM is found, the cloudlet is migrated to the most underutilized VM to ensure execution while minimizing system imbalance. This methodology makes the cloudlets scheduled, even if the deadlines are missed. This approach combines deadline awareness with load balancing, using migration to optimize both performance and VM utilization. The utilization of a computed deadline for each cloudlet and its reliance on the scaling factor k are the limitations of this algorithm. For instance, a lower k value assigns tighter deadlines, causing many cloudlets to miss their deadlines, while higher k values reduce the likelihood of missed deadlines by allowing more time for cloudlet completion. The k value is fixed as 2, which gives a maximum deadline for each cloudlet and creates an illusion that the cloudlets are meeting their deadlines. The migration of cloudlets among the VMs increases the scheduling cost and time. The algorithm will comply with the deadline for k=2 or more and produce fair VM utilization and load balancing [39].

The DBS algorithm considers both deadline and budget constraints. It categorizes cloudlets into three priority levels: high, fair, and low. High-priority cloudlets must satisfy both deadline and budget constraints, whereas fair-priority cloudlets prioritize only deadlines, and low-priority cloudlets prioritize only budgets. Each category of cloudlets is scheduled to the VMs in Cluster 1, Cluster 2, or Cluster 3. During allocation, the algorithm selects VMs capable of meeting the deadline, budget, or both constraints based on completion time and data transfer delay. The cloudlet is assigned to the VM with the earliest completion time among the eligible VMs. If no suitable VM is available, the cloudlet is rejected, which directly increases the number of deadlines missed and the overall cost. Resource utilization and load balancing also remain as challenges for the DBS algorithm [40].

The RDLBS2 methodology operates in two key phases. Initially it performs deadline-based allocation, where cloudlets are assigned to VMs based on their Expected Finish Time (EFT) to ensure deadlines are met. This allocation is carried out using schedulers such as Min-Min or Round Robin. In the second phase, RDLBS2 dynamically rebalances cloudlets using a receiver-initiated approach, where underloaded VMs pull cloudlets from overloaded ones. The α-conditioned migration ensures that cloudlets are only migrated if the target VM offers a significant performance improvement, as determined by the parameter α. This mechanism helps minimize penalties for missed deadlines and may improve turnaround time and VM utilization. However, the value of α critically influences performance: if α is too low (e.g., α = 0.1), very few migrations occur, potentially increasing deadline misses. Whereas if α is too high (e.g., α = 0.5), excessive migrations can increase overhead without effectively reducing penalties [41].

3. Problem Modeling and Proposed Framework

A. Problem Definition

In cloud environments, especially in time-critical applications, ensuring timely cloudlet execution is vital. Scheduling cloudlets with varying arrival times and deadlines onto available VMs is a complex and challenging task. Existing scheduling policies often result in deadline violations and suboptimal resource usage. TSDACS addresses these issues by introducing a deadline-aware scheduling approach that efficiently allocates the cloudlets to the available VMs and dynamically scales them within a scaling threshold (sMax). If the existing VMs cannot meet the deadlines of the cloudlets, scaling takes place to prevent missing deadlines. Horizontal scaling instantiates the new VMs. TSDACS only permits scaling up to a fixed number of times to control operational expenses and resource overuse. TSDACS efficiently assigns cloudlets to heterogeneous VMs in a way that minimizes deadline violations, makespan, and operational cost within acceptable limits of scaling.

The primary objective of TSDACS is to minimize:

1. **Deadline miss ratio:** The number of cloudlets missing their deadlines.

$$\min\left(\frac{1}{\text{cSize}} * \sum_{i=1}^{\text{cSize}} I.\left(fT(c_i) > D(c_i)\right)\right) \quad (1)$$

where fT(ci) and D(ci) are the finishing time and deadline of the cloudlet ci, respectively, and I is an indicator function that outputs 1 if the condition inside is true, 0 otherwise. cSize is the total number of cloudlets.

2. **Makespan:** The total time required to complete all cloudlets.

$$\min\left(\max_{i=1}^{\text{cSize}}(fT(c_i))\right) \quad (2)$$

Minimizes the makespan by finding a schedule that ensures the last cloudlet finishes as early as possible.

3. **Operational cost:** The expense associated with utilizing and scaling virtual machines.

$$\min\left(\sum_{i=1}^{\text{cSize}}\sum_{j=1}^{m+sMax} x_{ij}.\,bCost(vm_j).\left(\frac{L(c_i)}{S(vm_j)}\right) + \sum_{j=m+1}^{m+sMax} z_j.\,iCost(vm_j) + \sum_{j=m+1}^{m+sMax} y_j.\,sCost(vm_j)\right) \quad (3)$$

where xij∈ {0,1}, zj∈ {0,1}, and yj∈ {0,1} are all binary variables that indicate whether cloudlet ci is assigned to VM vmj, whether the VM is actively used, and whether the VM is a scaled VM. The operational cost includes both the VM usage costs comprising the base cost (bCost), the infrastructure cost (iCost), and the scaling cost (sCost). In this context, m and sMax represent the number of VMs instantiated during both stages, L(ci) denotes the length of the ith cloudlet, and S(vmj) indicates the speed of the j^{th} VM.

B. System Model

Cloudlets

A set of independent and non-pre-emptive cloudlets C={c1, c2, c3, . . ., ccSize} where each cloudlet ci is defined by:

Cloudlet Length L(ci) ∈ℝ+: The computational workload in million instructions (MI).

Arrival time A(ci) ∈ℝ+: The time at which the cloudlet arrives, in milliseconds.

Deadline D(ci) ∈ℝ+: The time by which the cloudlet must be completed, in milliseconds, such that

$$D(c_i) \geq A(c_i) + \frac{L(c_i)}{S(vm_{assigned})} \quad (4)$$

where S(vmassigned) is the speed of the VM to which the cloudlet is assigned.

Virtual Machines

TSDACS makes use of an initial set of virtual machines, VMinit={vm1, vm2, vm3, . . ., vmm}, and a set of scaled virtual machines, VMscaled={vmm+1, vmm+2, vmm+3, . . ., vmm+sMax}, making a total number of vmSize=m+sMax VMs, where each VM vmj has:

Processing speed S(vmj) ∈ℝ+: The processing speed of VM, measured in million instructions per second (MIPS).

Cost C(vmj) ∈ℝ+: The cost of using the VM per unit time, measured in dollars ($).

Scaling Constraint

A scaling limit (sMax) ∈ℤ+: The maximum number of virtual machines that can be scaled horizontally.

C. Problem Formulation

The minimum processing speed required for each cloudlet is determined using the formula (10), where tF(ci) is the timeframe or the window time within which the cloudlet must be completed to avoid deadline misses. The overhead time (oH) refers to the additional time associated with VM

provisioning delays, network latencies, etc. For experimental purposes, oH is fixed at 2.5 milliseconds. The VM extension factor (vmeF) is introduced to extend the VM speed beyond the actual required speed. The vmeF ensures the cloudlet meets its deadline even if a VM speed is slightly slower than expected. The vmeF ensures that VM performance remains stable and reliable despite real-time execution uncertainties. The vmeF is fixed as 1.15 throughout the experimentation.

$$\text{Total Cloudlet Length (totalLen)} = \sum_{i=1}^{cSize} L(c_i) \text{ in MI} \quad (5)$$

$$\text{Time Frame } tF(c_i) = D(c_i) - A(c_i) \text{ in msec} \quad (6)$$

$$\text{Maximum Time Frame (mtF)} = \text{Max}(tF(c_i)) \text{ in msec} \quad (7)$$

$$\text{Total VM Speed required (totalvmSpeed)} = \left\lceil \frac{\text{totalLen}}{\text{mtF}} \right\rceil \text{ in MIPS} \quad (8)$$

$$\text{Average VM Speed required (avgvmSpeed)} = \left\lceil \frac{\text{totalvmSpeed}}{\text{cSize}} \right\rceil \text{ in MIPS} \quad (9)$$

$$\text{Required Processing Speed sVM[i]} = \left\lceil \frac{L(c_i)}{(tF(c_i) - oH))} * \text{vmeF} \right\rceil \text{ in MIPS} \quad (10)$$

If vmeF>1, VM speeds are increased beyond the required speeds to prevent deadline misses.

If vmeF=1, VM speeds are allocated exactly as required, assuming ideal execution conditions.

If vmeF<1, VM speeds are allocated lower than the required speeds, which may result in missed deadlines due to insufficient capacity.

The per-unit time cost of a VM is based on its speed (S(vm_j)) relative to the average speed (avgvmSpeed) of all VMs. The base cost (bCost) represents the fundamental cost and is typically proportional to the computational capacity of the VM. It is influenced by factors such as the number of processing elements or CPU cores, clock speed, RAM size, storage type and size, and network bandwidth. The infrastructure cost (iCost) of a VM represents the fixed costs associated with running it, regardless of its exact performance. This includes the cost of physical servers in the data center, power consumption, maintenance, and administration costs. Scaling cost (sCost) is the monetary expense incurred when adding a new VM to the system to meet cloudlet deadlines.

$$\text{Cost of VM per unit time } C(vm_j) = \left(\left(\frac{S(vm_j)}{\text{avgvmSpeed}} \right) * \text{bCost} \right) + \text{iCost} + \text{sCost} \quad (11)$$

$$\text{Response Time } rT(c_i) = S(c_i) - A(c_i) \quad (12)$$

Response time (rT) is the time taken for a cloudlet to begin execution after its arrival in the system. A lower response time indicates better system efficiency. If S(ci)=A(ci), the task starts execution immediately. If S(ci)>A(ci), there is a delay due to scheduling, resource unavailability, or other factors. Here, S(ci) denotes the start time of the cloudlet ci.

$$\text{VM Utilization vmUt}(Vm_j) = \frac{bT(vm_j)}{msT} \quad (13)$$

$$\text{Average VM Utilization (avmUt)} = \left(\frac{1}{\text{vmSize}} * \sum_{j=1}^{\text{vmSize}} \frac{bT(vm_j)}{msT} \right) \quad (14)$$

$$\text{Makespan (msT)} = \text{Max}\left(\sum_{i=1}^{cSize} fT(c_i) \right) \quad (15)$$

Where vmSize is the total number of VMs utilized during the schedule, i.e., the number of VMs initially created and the number of scaled VMs (vmSize=m+sMax). The busy time of a VM, bT(vm), refers to the total amount of time a VM spends executing cloudlets. Makespan (msT) represents the maximum finishing time among all cloudlets. A lower msT indicates that all cloudlets complete execution more quickly, which in turn improves overall system throughput and resource efficiency.

$$\text{Load Imbalance Level (libL)} = \sqrt{\frac{1}{\text{vmSize}} * \sum_{j=1}^{m} \left(\text{vmUt}(vm_j) - \text{avmUt} \right)^2} \quad (16)$$

Load imbalance level (libL) computes the deviation of individual VM utilization from the average utilization. A lower libL value indicates better load balancing, depicting the cloudlets are distributed evenly across VMs.

$$\text{Profit} = \text{baseFee} - (\text{C}(\text{vm}_\text{j}) * \text{eT}(\text{c}_\text{i}) + (\text{penalty} * \text{lT}(\text{c}_\text{i})) \quad (17)$$

$$\text{Loss} = (\text{C}(\text{vm}_\text{j}) * \text{eT}(\text{c}_\text{i}) + (\text{penalty} * \text{lT}(\text{c}_\text{i})) \quad (18)$$

$$\text{lT}(\text{c}_\text{i}) = \begin{cases} 0, & \text{fT}(\text{c}_\text{i}) \leq \text{D}(\text{c}_\text{i}) \\ \text{fT}(\text{c}_\text{i}) - \text{D}(\text{c}_\text{i}), & \text{fT}(\text{c}_\text{i}) > \text{D}(\text{c}_\text{i}) \end{cases} \quad (19)$$

$$\text{Total Gain} = \text{Max}(\text{Profit} - \text{Loss}, 0) \quad (20)$$

$$\text{Total Loss} = \text{Max}(\text{Loss} - \text{Profit}, 0) \quad (21)$$

Profit represents the total earnings of a cloud service provider from executing a cloudlet. It consists of a fixed base fee (revenue), minus the cost of VM execution time and any penalty for late completion. Loss accounts for the total cost incurred, including the VM usage cost and the penalty due to late execution. A higher profit and lower loss indicate efficient execution with minimal delays. Where base Fee is the fixed charge for cloudlet execution, C(vmj) is the per-unit time cost of a VM as depicted in (11), penalty is the cost deducted for late completion of the cloudlet, and lT is the late time. Total Gain and Total Loss measure the overall financial gain of executing the cloudlets in a cloud environment and are based on profit and loss.

If Profit is greater than Loss, then Total Gain=Profit-Loss. Otherwise, Total Gain = 0.

If Loss is greater than Profit, then Total Loss=Loss-Profit. Otherwise, Total Loss = 0.

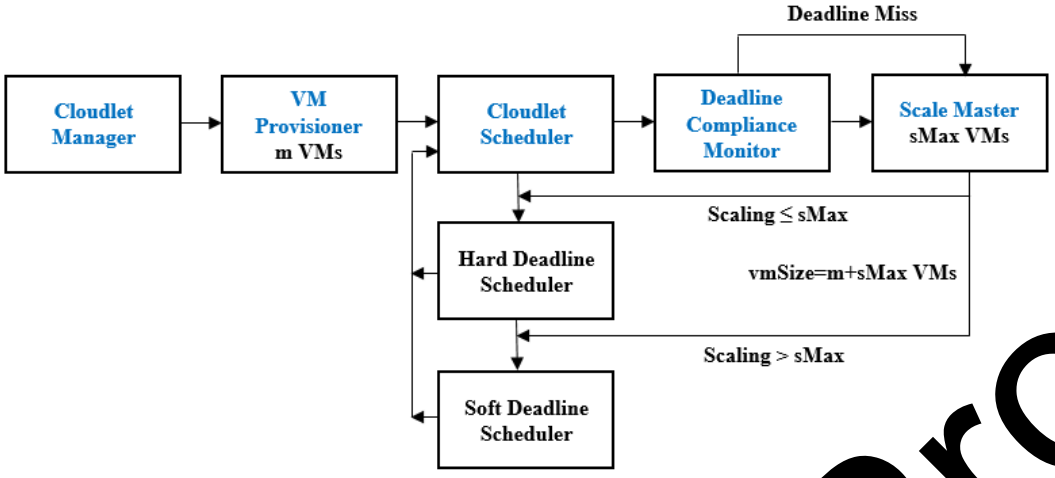
$$\text{Deadline Miss Ratio (dmR)} = \left(\frac{1}{\text{cSize}} * \sum_{\text{i}=1}^{\text{cSize}} \text{I}.(\text{fT}(\text{c}_\text{i}) > \text{D}(\text{c}_\text{i}))\right) * 100 \quad (22)$$

The deadline miss ratio (dmR) measures the proportion of cloudlets that miss their deadlines out of the total number of cloudlets.

D. Proposed Framework

The framework of the TSDACS algorithm is illustrated in Figure 1. TSDACS comprises five major functional components. The cloudlet manager handles the cloudlets arriving at different intervals and prioritizes them based on their respective deadlines. The VM provisioner module calculates the initial minimum number of VMs required for executing the cloudlets while ensuring deadline compliance. Initially, it provisions a set (vmList) of VMs with m instances. The cloudlet scheduler is a core component of TSDACS, responsible for assigning cloudlets to the initially provisioned m VMs in a manner that avoids deadline violations. It contains two sub-modules: the soft deadline scheduler and the hard deadline scheduler. Although the VMs set up in stage 1 are initially estimated to be sufficient for executing cloudlets within their deadlines, unpredictable execution dynamics, such as runtime delays, workload spikes, or VM load and performance variability, may lead to deadline violations. Stage 2 addresses this issue by dynamically scaling the system, provisioning additional VMs as needed to maintain deadline compliance. The deadline compliance monitor oversees the system for any deadline violations. Upon detecting a deadline miss, it triggers the scale master module. The scale master horizontally scales the system by provisioning an additional VM. However, the number of scalings is restricted to a maximum of sMax times. Thus, the maximum number of VMs in the system becomes m+sMax. Cloudlets continue to be scheduled using the hard deadline scheduler module as long as the deadlines are met. If deadline compliance cannot be maintained even after reaching the maximum VM limit, the cloudlets are scheduled using the soft deadline scheduler, which assigns cloudlets based on the earliest finishing time among VMs. This method of cloudlet execution, even after deadlines are missed using the soft deadline scheduler, ensures cloudlet completion and reduces overall delay, preserving system responsiveness and user satisfaction. Thereby, TSDACS avoids the over-provisioning of resources, ensures the deadline of cloudlets, and maintains the overall performance.

Figure 1: Proposed TSDACS Framework



4. Proposed Methodology

The TSDACS algorithm is a deadline-aware cloudlet scheduling algorithm designed for deadline-sensitive cloud environments. It focuses on minimizing deadline violations and makespan while ensuring optimal cost management by efficiently allocating cloudlets to the available VMs. The algorithm also aims to reduce response time and improve VM utilization and load balancing. TSDACS operates in two stages. In the first stage, the optimal number of VMs is provisioned based on cloudlet requirements, considering factors such as VM processing speed, RAM, storage, and bandwidth. Instead of provisioning randomly configured VMs, the algorithm estimates cloudlet demands and provisions suitable VMs accordingly. This targeted approach enhances deadline compliance. However, even with appropriately provisioned VMs, cloudlets may still miss deadlines due to overheads such as scheduling delays, resource contention, high VM load, etc.

Task migration, over-provisioning of resources, resource scaling, and soft deadline handling are commonly used techniques to manage deadlines in time-sensitive applications. However, task migration and over-provisioning can be costly and may introduce execution delays and resource wastage due to their associated overheads. To address deadline misses, TSDACS employs dynamic horizontal VM scaling to provision additional VMs as needed and soft deadline handling to improve the likelihood of meeting deadlines. The second stage involves adding a limited number of VMs to improve deadline compliance. The number of additional VMs is limited by an estimated threshold to avoid over-provisioning, control costs, reduce infrastructure load, optimize resource utilization, and ensure quality service delivery. TSDACS also applies a soft allocation strategy by assigning cloudlets to VMs with the earliest finishing time, thereby further reducing overall delay and mitigating execution slowdowns. Algorithm 4.1 outlines the overall functionality of the TSDACS algorithm, while Algorithms 4.2 to 4.5 detail the individual operations involved.

Algorithm 4.1: Two-Stage Deadline-Aware Cloudlet Scheduler (TSDACS)

Input:

C←Set of cloudlets ci = (L(ci), A(ci), D(ci)), ∀ i ∈ {1, 2, ..., cSize}.

Output:

An optimized cloudlet-VM mapping with minimal deadline misses (dM), response time (rT), makespan (msT), and load imbalance level (libL), along with improved virtual machine utilization (vmUt), and profit/loss.

Begin TSDACSMapper

1. Sort cloudlets:

Sort cloudlets in C by their deadlines D(ci) in ascending order.

2. Estimate required VMs-vmEstimator ()

Compute the minimum processing speed required for each cloudlet and store it in sVM[].

Sort sVM [] in descending order.

Select m VMs from sVM [] for initial provisioning.

Calculate operational cost per unit time $C(vm_j)$ for each VM.

Create a set of VMs where VM= {vmj | vmj=($S(vm_j)$, $C(vm_j)$), $\forall j \in \{1, 2, ..., m\}$}.

Set the upper threshold for VM scaling (sMax).

3. Initialize execution time matrix-initexeMatrix ()

Compute execution time estimates for all cloudlet-VM pairs.

4. Schedule cloudlets using findBestVM (), scale Master (), softDeadlineScheduler (), and submit Cloudlet ()

For each cloudlet ci∈ C:

Find the best VM that meets the deadline for ci by invoking findBestVM(ci).

If no suitable VM is found (vmId==-1):

Check if scaling is allowed (scaling≤sMax):

If scaling is allowed, invoke scaleMaster(estF(ci)) to provision a new VM.

Calculate cost and characteristic for the new VM (steps 2d and 2e).

Re-invoke findBestVM(ci) to set the new VM as the best VM.

If scaling is not allowed (scaling>sMax), then

Invoke softDeadlineScheduler () to assign ci to the VM with the earliest finishing times.

Assign ci to the best VM using submit Cloudlet ().

Repeat 4a to 4c until all cloudlets are scheduled.

5. Compute performance metrics-performanceEstimates ()

Compute dM, rT, wsT, vmUt, libL, and profit/loss.

6. Print final schedule-printCloudlets ()

Output the cloudlet-VM mapping and performance summary.

End TSDACSMapper

VM Configuration Estimation and Generation: In this stage, the cloudlets are first sorted by their deadlines, giving higher priority to cloudlets with tighter deadlines to ensure timely execution. The algorithm then invokes vmGenerator () to determine the ideal number of VMs required for executing the cloudlets, denoted as m. The algorithm calculates the minimum processing speed required for each cloudlet and stores it in sVM [], using formula (10). The estimated VM speeds are then sorted in descending order to prioritize assigning higher-speed VMs to cloudlets with more critical deadlines. Next, a subset of VMs is selected based on the total estimated computational requirement, denoted as totalvmSpeed. The minimum number of VMs (m) is determined by cumulatively selecting elements

from sVM [] until their sum meets or exceeds totalvmSpeed (i.e., sum(sVM[])≥totalvmSpeed). Each selected VM is then allocated computational resources and assigned an operational cost according to its processing speed. To manage scalability, a maximum scaling threshold (sMax) is defined. This threshold limits the number of VM scaling operations based on workload distribution, thereby avoiding excessive resource provisioning that could overutilize the data center infrastructure in real-time scenarios. The workload balance (wB) is an adaptive parameter that regulates how much the system can scale VMs dynamically based on the workload-to-VM ratio. It helps balance performance, cost, and resource utilization during times when the execution dynamics of the cloudlets are unpredictable. This ensures limited scalability and deadline compliance. The vmGenerator () algorithm is depicted in 4.2.

Algorithm 4.2: vmGenerator ()

Input:

C←Set of cloudlets ci = (L(ci), A(ci), D(ci)), ∀ i ∈ {1, 2, ..., cSize}.

Output:

m←Number of required VMs.

vm[m]←Array of configured VMs.

Begin vmEstimator ()

for each cloudlet ci in C:

totalLen= totalLen+L(ci)

tF(ci)←D(ci)-A(ci)

mtF←max (mtF, tF(ci))

end for

totalvmSpeed←ceil(totalLen/mtF)

avgvmSpeed←ceil(totalvmSpeed/cSize)

Compute the minimum processing speed required sVM []:

for each cloudlet ci in C:

sVM[i]←ceil(L(ci)/(tF(ci)*wH)*vmeF)

end for

Sort sVM [] in ascending order

Select m VMs from sVM[]:

m←0

cumSpeed←0

for i=0 to cSize-1 do

cumSpeed←cumSpeed+sVM[i]

m←m+1

if cumSpeed≥totalVMSpeed then

break

end if

end for

Initialize parameters: nPes, ram, bw, size, vmm, iCost, bCost.

Calculate the VM operational cost and create VMs:

for each VM vmj in m:

$C(vm_j) \leftarrow ((sVM[j]/avgvmSpeed) * bCost) + iCost$

$vm[j] \leftarrow vm.add(id=j, S(vmj)=sVM[j], nPes, ram, bw, size, vmm, cost=C(vm_j))$

end for

Define the upper threshold for VM scaling:

Compute VM scaling threshold (sMax) for limiting the number of VM scaling.

$sMax \leftarrow floor((cSize/vmSize)*wB)$

End

Cloudlet Execution and Performance Evaluation: Once the VMs are provisioned, cloudlet scheduling and execution take place. The initexeMatrix() function is invoked to compute the expected execution times of cloudlets across the available VMs, allowing for informed decision-making. Cloudlets are then scheduled using the findBestVM(), scaleMaster(), softDeadlineScheduler (), and submitCloudlet() functions. The findBestVM () function determines the most suitable VM for each cloudlet based on execution time and deadline constraints. Cloudlets are always scheduled to VMs with the minimum completion time that meets the deadline. However, if no suitable VM is available, the algorithm checks whether additional VMs can be created within the predefined maximum scaling threshold (sMax). If scaling is possible, the scaleMaster() function is invoked to dynamically provision a new VM to handle the potential deadline miss. The cloudlet is then assigned to either an existing or a newly created VM, ensuring execution proceeds without violating the deadline. If a new VM cannot be provisioned or no suitable VM is available, the cloudlet(s) are allocated to the VM with the earliest completion time using softDeadlineScheduler (). This process continues until all cloudlets have been successfully scheduled using the submit Cloudlet () method, supported by the TSDACSMapper (). Once execution is completed, the algorithm evaluates key performance metrics to assess scheduling efficiency. The performance Estimates () function computes the dM, rT, msT, vmUt, libL, and profit. Finally, the print Cloudlets () function outputs the cloudlet-VM mapping and execution details.

Algorithm 4.3 findBestVM ()

Input:

x←Index of the cloudlet to be scheduled.

vm[]←List of VMs provisioned.

exeMat[][]←Execution time matrix.

sMax←Maximum number VMs that can be scaled.

scaling←Current number of VMs that have been scaled.

Output:

vmId→ID of the selected VM for execution.

Begin findBestVM ()

Initialize variables: min←∞, vmId←-1, c←C[x].

Find the best available VM:

for j=0 to vm.size()-1 do

if (exeMat[x][j] <min and exeMat[x][j] <=D(x)) then

min←exeMat[x][j]

vmId←j

end if

end for

Handle missed deadline through scaling:

if (vmId== -1 and scaling Smax) then

vmId←scaleMaster (c, tF(c))

end if

Handle missed deadline by allocating cloudlet c to the VM with earliest finishing time:

if (vmId==-1 and scaling>sMax) then

for j=0 to vm. size ()-1 do

if (exeMat[x][j] <min) then

min←exeMat[x][j]

vmId←j

end if

end for

end if

upgrade exeMatrix()

return vmId

end

Algorithm 4.4: scale Master(ci, tF(ci))

Input:

L(ci)←Cloudlet length.

tF(ci)←Time frame for execution.

Output:

vmId→ID of the newly created VM or -1 if scaling fails.

Begin scaleMaster(ci, tF(ci))

Initialize VM parameters: nPes, ram, bw, size, vmm, iCost, bCost, sCost.

Compute VM speed, cost and create a new VM:

sVM[j]←ceil(L(ci)/(tF(ci)-oH)*vmeF)

C(vm_j)←((sVM[j]/avgvmSpeed) *bCost) +iCost+sCost

Check for total available MIPS:

if (taMIPS<S(vmj)) then

Print "Insufficient Resource!!"

return -1

end if

else

vm[j]←vm.add(id=j, S(vmj)=sVM[j], nPes, ram, bw, size, vmm, cost=C(vmj))

scaling←scaling+1

end else

End

Algorithm 4.5: submit Cloudlet()

Input:

C←Set of cloudlets ci = (L(ci), A(ci), D(ci)), ∀ i ∈ {1, 2, ..., cSize}.

Output:

exeC→List of executed cloudlets.

cloudletMap→Map storing execution details (start time, finish time, VM ID, status, etc.).

Begin submitCloudlet ()

Call TSDACSMapper(C)

Log the execution details of each cloudlet into cloudletMap.

End

Time Complexity Analysis

The efficiency and scalability of any scheduling algorithm largely depend on its time complexity. The CPDALB, DBS, and TSDACS algorithms have a time complexity of O(n2m), indicating that their runtime grows quadratically with the number of cloudlets n and linearly with the number of virtual machines m. In contrast, the RDLBS2 algorithm has a time complexity of O(nm), which grows linearly with the product of n and m, making it more efficient in terms of computational overhead. From a time complexity perspective, RDLBS2 outperforms CPDALB, DBS, and TSDACS. However, when considering performance metrics such as the number of deadline misses, makespan, and cost, RDLBS2 lags behind CPDALB and TSDACS. Even though TSDACS incurs higher computational overhead due to its enhanced cloudlet-to-VM mapping, dynamic VM scaling, and adaptive soft allocation techniques, this overhead leads to faster cloudlet execution and fewer missed deadlines. Additionally, its ability to reduce VM over-provisioning makes TSDACS particularly effective in environments where meeting deadlines and resource constraints are critical.

5. Experimental Setup

We have evaluated the proposed TSDACS algorithm through simulation using the Java-based Cloud Sim 3.0.3 toolkit, a simulator for modelling and evaluating cloud infrastructures. The simulation environment of TSDACS consists of a single data center with 5 to 15 host instances. Hosts use the VmSchedulerSpaceShared policy and support dual-core or quad-core processors with a bandwidth of 10000 Mbps. The system operates on the Linux operating system with Xen as the hypervisor and a maximum host memory capacity of 100000 MB each. Cloudlets are scheduled using CloudletSchedulerSpaceShared, with lengths ranging from 10000 to 75000000 MI, 6 to 200 cloudlets in total, and 1 processing element (PE) per cloudlet. We compute the number of VMs and their configuration based on the length of the cloudlet, the arrival time, and the deadline constraint. The specifications of VMs used in the experimental evaluation generally vary from 2 to 50 VMs, with each configured with 1 PE, 512 MB of memory, and a bandwidth of 1000 Mbps. VM speeds range between 5000 and 250000 MIPS. The simulator was run on Windows 10 with an AMD PRO A4-4350B R4 (2 CPU + 3 GPU cores, 2.50 GHz), 4 GB of RAM, and a 64-bit x64-based processor.

Due to the structural constraints of the Cloud Sim simulator, TSDACS uses horizontal scaling instead of vertical scaling. The scheduling models in Cloud Sim, such as VmScheduler and Cloudlet Scheduler, operate based on the assumption of fixed resource capacities for VMs and fixed requirements for cloudlets. The configuration of a VM, including CPU speed, RAM, and bandwidth, is determined at the time of VM creation and remains constant throughout the simulation. Similarly, the requirements of a cloudlet, such as the number of PEs and its expected CPU, RAM, and bandwidth usage, are set during its creation and do not change during execution. Vertical scaling involves the dynamic adjustment of VM resources or configurations during execution. cloud Sim 3.0 does not support these characteristics. As a result, TSDACS adopts a controlled horizontal scaling approach, where additional VMs can be provisioned as needed to meet workload deadline demands. The performance measures used to evaluate the TSDACS algorithm include makespan, profit, loss, total gain, total loss, response time, VM utilization ratio, load imbalance level, scaling limits, deadline misses, and deadline miss ratio.

6. Results and Discussion

This section presents and analyses the experimental results of the proposed TSDACS algorithm in comparison with three existing deadline-aware scheduling algorithms: CPDALB, DBS, and RDLBS2. The evaluation focuses on key performance metrics, including deadline compliance, makespan, response time, VM utilization, load balancing, and cost benefits. Each metric is discussed in detail to highlight the effectiveness and practical advantages of TSDACS in deadline-sensitive cloud computing environments.

A. Results

Table 3 demonstrates the performance of the TSDACS algorithm under varying workload conditions, emphasizing its flexibility in efficient VM management, attaining deadlines, and maximizing profitability. The table includes the number of VMs provisioned during both the initial allocation and the scaling stages. Key metrics reported are makespan (msT), profit and loss, average response time (arT), average VM utilization (avmUt), load imbalance level (libL), and the number of deadlines misses after scaling. Tables 4, 5, and 6 present the performance of the CPDALB, DBS, and RDLBS2 algorithms, respectively, while Tables 7 and 8 provide a consolidated comparison of all four algorithms.

Table 3: Performance of TSDACS Algorithm

S. No.	No. of Cloudlets	Total No. of VMs	Makespan (msT) in msec	Profit in $	Loss in $	Average Response Time (arT) in msec	Average VM Utilization (avmUt)	Load Imbalance Level (libL)	No. of Deadline Misses
1	6	2	107.22	$35.53	$3.75	17.08	0.8	0.01	1
2	10	4	153.66	$59.72	$9.79	25.42	0.65	0.04	1
3	15	5	172.53	$89.68	$16.19	28.55	0.66	0.15	2
4	24	11	462.81	$149.50	$64.78	44.77	0.59	0.24	1
5	37	13	760.16	$237.09	$154.56	96.64	0.62	0.1	3
6	46	15	836.49	$299.09	$214.37	103	0.64	0.21	[illegible]
7	57	9	790.27	$356.67	$165.02	118.79	0.85	0.08	4
8	66	10	396.35	$397.30	$94.58	50.44	0.87	0.0[illegible]	5
9	75	5	972.71	$501.94	$551.53	406.57	0.96	0.[illegible]	3
10	84	6	747.62	$564.09	$614.33	318.47	0.65	0.2[illegible]	6
11	97	6	1457.66	$649.72	$628.88	358.09	0.[illegible]5	0.02	7
12	105	42	1384.63	$714.88	$692.69	77.45	[illegible]66	0.15	0
13	120	35	3833.17	$949.17	$1,699.8	296.34	0.[illegible]	0.19	6
14	142	16	2546.63	$968.44	$1,055.9	255.98	0.73	0.12	9
15	157	18	7085.89	$1,014.9	$2,963.9	776.8	0.5[illegible]	0.09	5
16	166	12	2405.78	$465.08	$2,992.4	890.69	[illegible]85	0.05	12
17	173	10	2674.13	$589.77	$3,128.1	902[illegible]98	0.86	0.05	14
18	188	15	2571.03	$692.03	$3,787.8	97[illegible]82	[illegible].82	0.08	11
19	192	23	1858.88	$385.59	$1,738.0	211[illegible]	[illegible].45	0.11	0
20	200	29	4436.79	$1,682.2	$4[illegible]80.8	[illegible]1127.66	0.86	0.06	10

Table:4 Perfor[illegible]nce of C[illegible]DLAB [illegible]gorithm

S. No.	No. of Cloudlets	Initial No. of VMs (Stage: 1)	Total [illegible] of VMs [illegible] Scaling (Stage:2)	Makes[illegible]pan (msT) in msec	Profit in $	Loss in $	Average Response Time (arT) in msec	Average VM Utilization (avmUt)	Load Imbalance Level (libL)	No. of Deadline Misses after Scaling (Stage:2)
1	6	2	2	90.28	$35.2	$1.67	8.08	0.75	0.07	0
2	10	3	4	140.68	$59.0	$3.69	16.42	0.58	0.02	0
3	[illegible]	[illegible]	5	161.02	$88.4	$6.16	19.55	0.67	0.2	0
4	24	10	11	453.57	$145.	$32.3	35.77	0.55	0.18	0
5	[illegible]7	[illegible]12	13	741.1	$226.	$67.7	87.64	0.59	0.26	2
6	4[illegible]	14	15	821.4	$282.	$91.0	94	0.63	0.13	0
7	5[illegible]	7	9	776.01	$343.	$54.1	109.79	0.79	0.04	0
8	66	8	10	379.77	$389.	$31.4	41.44	0.8	0.09	0
9	75	3	5	956.66	$444.	$55.5	397.57	0.78	0.05	1
10	84	4	6	729.24	$495.	$30.7	309.47	0.59	0.2	0
11	97	6	6	1373.4	$579.	$92.8	349.09	0.95	0.06	0
12	105	41	42	1325.5	$676.	$426.	68.45	0.66	0.18	6
13	120	34	35	3774.6	$843.	$964.	287.34	0.67	0.14	5
14	142	9	16	2491.3	$878.	$429.	246.98	0.72	0.1	3
15	157	15	18	7011.77	$859.16	$902.33	767.8	0.55	0.12	2

16	166	10	12	2354	$235.52	$325.21	881.69	0.65	0.03	2
17	173	9	10	2626.8	$235.	$276.	893.98	0.65	0.07	1
18	188	13	15	2519.6	$492.	$409.	966.82	0.71	0.11	0
19	192	20	23	1794	$368.	$330.	202.41	0.57	0.09	0
20	200	26	29	4361.9	$1,35	$1,20	1118.66	0.74	0.04	2

Table 5: Performance of DBS Algorithm

S. No.	No. of Cloudlets	Total No. of VMs	Makespan (msT) in msec	Profit in $	Loss in $	Average Response Time (arT) in msec	Average VM Utilization (avmUt) in msec	Load Imbalance Level (libL)	No. of Deadline Misses
1	6	2	93.76	$36.84	$7.24	16.74	0.73	0.[illegible]	0
2	10	4	139.99	$120.65	$40.67	16.57	0.61	[illegible]	13
3	15	5	155.49	$66.94	$20.62	17.69	0.[illegible]	0.1[illegible]	12
4	24	11	452.84	$157.33	$29.32	47.91	[illegible].57	0.18	11
5	37	13	779.71	$298.08	$132.71	95.08	0.[illegible]6	0.18	1
6	46	15	951.16	$310.37	$205.42	101.06	0.55	0.17	0
7	57	9	741.07	$375.83	$167.67	129.66	0.74	0.14	5
8	66	10	318.36	$362.41	$64.94	50.67	[illegible].75	0.11	15
9	75	5	1168.82	$457.65	$536.28	36[illegible]8	0.77	0.15	5
10	84	6	1105.43	$504.73	$651.12	35[illegible].16	[illegible]0.45	0.04	17
11	97	6	1409.11	$682.71	$595.61	370[illegible]6	0.88	0.05	7
12	105	42	1455.04	$748.33	$[illegible]57.51	156.41	0.57	0.16	8
13	120	35	3788.76	$980.14	$1,67[illegible]3	[illegible]9.78	0.59	0.19	2
14	142	16	2429.74	$964.0[illegible]	$1,03[illegible]6	23[illegible]83	0.71	0.13	6
15	157	18	7746.35	$1,062.92	$2[illegible]6.45	1205	0.48	0.21	10
16	166	12	2893.51	$426.37	$[illegible]006.89	867.99	0.64	0.03	0
17	173	10	2957.33	$642.37	$3,1[illegible]40	878.84	0.61	0.04	3
18	188	15	3322.56	$[illegible]3.24	$3,754.97	916.78	0.55	0.05	1
19	192	23	1845.23	$429.[illegible]	$1,764.78	246.35	0.4	0.07	7
20	200	29	4839.1[illegible]	[illegible]962.8[illegible]	$4,348.94	1193.43	0.71	0.16	10

Table 6: Performance of RDLBS2 Algorithm

S. No.	No. of Cloudlets	Total No. of VMs	Makespan (msT) in msec	Profit in $	Loss in $	Average Response Time (arT) in msec	Average VM Utilization (avmUt) in msec	Load Imbalance Level (libL)	No. of Deadline Misses
1	6	2	98.6	$35.53	$4.13	35.02	0.75	0.12	1
2	10	4	159.04	$59.72	$9.49	43.76	0.59	0.16	1
3	15	5	165.18	$89.60	$15.37	13.47	0.6	0.12	2
4	24	11	497.77	$149.52	$66.99	59.57	0.51	0.17	1
5	37	13	806.44	$237.13	$154.13	115.35	0.63	0.18	4
6	46	15	904.57	$299.18	$215.34	102.87	0.56	0.13	5
7	57	9	708.82	$357.73	$177.76	106.51	0.75	0.14	6
8	66	10	311.59	$397.72	$98.71	66.83	0.84	0.08	7
9	75	5	1171.89	$511.43	$506.03	387.63	0.74	0.04	2
10	84	6	1101.17	$568.11	$652.95	319.93	0.52	0.01	13
11	97	6	1351.88	$659.76	$641.96	363.18	0.91	0.03	0
12	105	42	1311.14	$727.77	$798.89	91.83	0.68	0.14	8
13	120	35	4085.07	$941.88	$1,648.1	294.45	0.61	0.25	9

14	142	16	2508.73	$965.64	$1,033.3	243.94	0.69	0.15	10
15	157	18	6926.72	$1,085.8	$3,855.2	790.31	0.59	0.25	11
16	166	12	2972.15	$463.08	$2,955.8	876.08	0.61	0.03	12
17	173	10	2885.59	$574.12	$2,908.5	897.35	0.68	0.05	13
18	188	15	3184.48	$682.08	$3,661.9	974.43	0.64	0.03	15
19	192	23	1686.05	$399.00	$1,831.3	215.13	0.51	0.02	0
20	200	29	4796.19	$1,676.1	$4,459.5	1145.74	0.71	0.18	17

Table 7: Consolidated Key Performance Indicator Table-1

S. No.	Algorithm(s)	Average Makespan (amsT) in msec	Total Profit in $	Total Loss in $	Total Gain in $	Total Loss in $
1	CPDALB	1782.7205	$10,802.49	$24,957.54	$944.57	$15,099.62
2	DBS	1929.6695	$10,962.86	$24,801.63	$1,217.78	$13,956.55
3	RDLBS2	1881.6535	$10,880.99	$25,695.64	$907.41	$15,721.06
4	TSDACS	1744.15	$9,034.57	$5,733.79	$3,195.50	$294.82

Table 8: Consolidated Key Performance Indicator Table-2

S. No.	Algorithm(s)	Average Response Time (arT) in msec	Average VM utilization (avmUt)	Average Load Imbalance Level (alibL)	No. of Deadline Misses (dM)	Deadline Miss Ratio (dmR) %
1	CPDALB	354.1475	0.73	0.103	105	5.36
2	DBS	375.7065	0.68	0.1215	133	6.79
3	RDLBS2	357.169	0.656	0.114	137	6.99
4	TSDACS	345.1475	0.78	0.109	24	1.22

B. Discussion

6.2.1 Makespan (msT): Figure 2 depicts the average makespan (amsT) performance of the four algorithms. A good scheduling algorithm should result in a minimal makespan, or overall completion time. The experimental evaluation demonstrates that the performance of TSDACS surpasses CPDALB, DBS, and RDLBS2 by 2.16%, 9.62%, and 7.31%, respectively. The makespan metric is indirectly associated with resource efficiency, cost savings, and the timely execution of workloads. Even a small reduction in makespan leads to substantial savings, better resource utilization, and a higher quality of service. In deadline-sensitive cloud environments, an ideal algorithm must balance minimizing deadlines while minimizing makespan, and the results indicate that TSDACS is highly suitable for such environments.

6.2.2 Response Time (rT): TSDACS also outperforms the other three algorithms in terms of response time (rT), showing reductions of 2.45%, 8.135%, and 3.37%. Similar to makespan reduction, minimizing response time plays a significant role in resource efficiency, cost savings, and the timely execution of workloads. Figure 3 illustrates the average response time across all four algorithms.

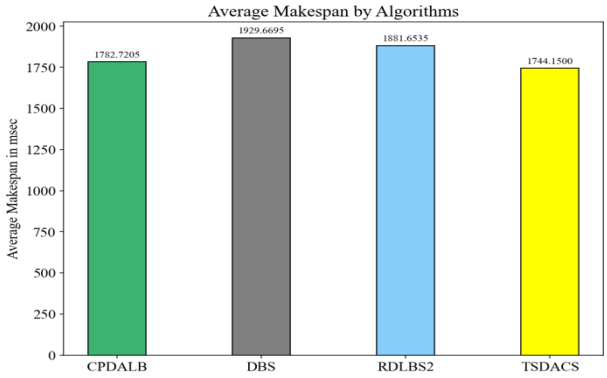


Figure 2: Makespan Comparison

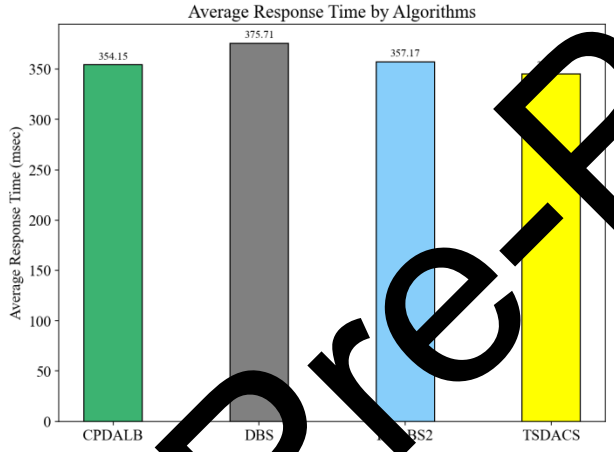


Figure 3: Response Time Comparison

6.2.3 Virtual Machine Utilization and Load Imbalance Level (vmUt & libL): Figure 4 compares four algorithms based on average VM utilization (avmUt) and average load imbalance level (alibL). CPDALB achieves the highest utilization (0.733) and the lowest imbalance (0.103), while DBS shows the lowest utilization (0.628) and a slightly higher imbalance (0.121). RDLBS2 and TSDACS exhibit moderate utilization of 0.650 and 0.680, with load imbalance levels of 0.114 and 0.109, respectively. Although CPDALB outperforms TSDACS in terms of average VM utilization and load imbalance level, it does not achieve optimal performance in meeting deadlines. In deadline-sensitive environments, meeting deadlines takes priority over maximizing VM utilization or load balancing.

6.2.4 Total Gain and Total Loss: Figure 5 shows the total gain and total loss performance of all four algorithms. CPDALB, DBS, and RDLBS2 have high losses between $15,000 and $15,700 and gains under $4,500. In contrast, TSDACS has the highest gain of $3,595.60 and the lowest loss of $294.82, showing much better financial results. This demonstrates that TSDACS is a good choice for cloud service providers and users working in critical, time-sensitive environments. TSDACS achieves about 66% to 75% more gain than the other algorithms and reduces losses by about 98%, almost eliminating losses compared to the other three algorithms.

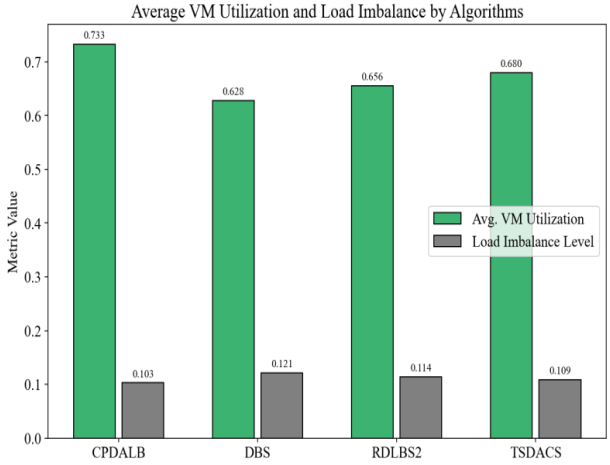


Figure 4: VM utilization and Load Imbalance Comparis

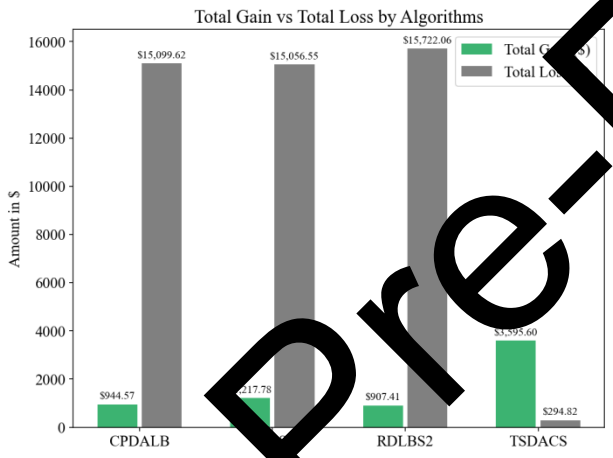


Figure 5: Total Gain a Total Loss Comparison

6.2.5 Deadline Misses and Deadli Miss Ratio (dM & dmR): The two Figures 6 & 7, compare the deadline misses (dM) and deadli mi o (dmR) of four algorithms. TSDACS demonstrates the best performance with the f west deadlin misses (24) and the lowest miss ratio (1.22%), while RDLBS2 records the high number f sses (137) and the highest miss ratio (6.99%). CPDALB and DBS result in deadline m s ratios of 5.36% and 6.79%, respectively. Overall, TSDACS shows superior efficiency i nee g de lines compared to the other algorithms. The lower deadline misses of TSDACS depict hat it is h hly effective for deadline-sensitive environments and suitable for time-critical application

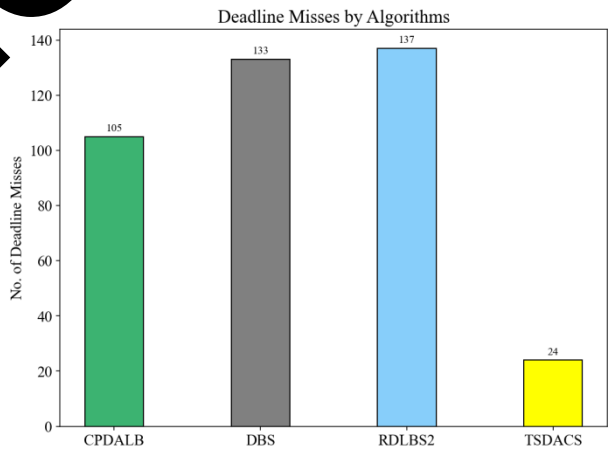


Figure 6: Deadline Miss Comparison

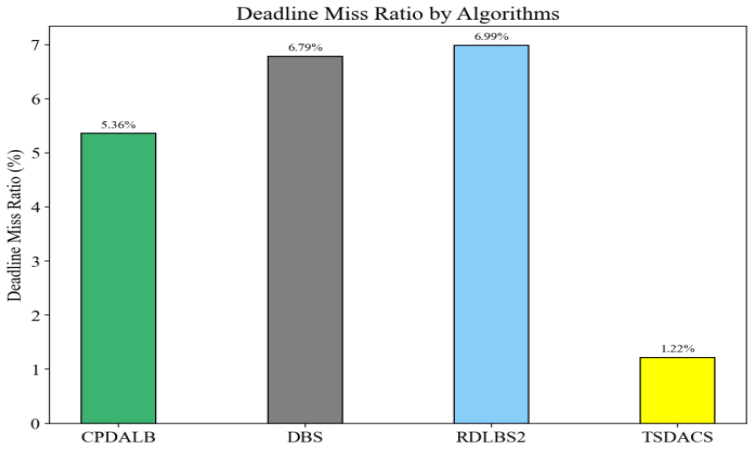


Figure 7: Deadline Miss Ratio Comparison

7. Conclusion

The TSDACS algorithm offers an effective solution for deadline-sensitive cloudlet scheduling through a two-stage optimization process: initial VM provisioning based on cloudlet requirements, followed by runtime scaling that is horizontal and controlled, along with soft cloudlet allocation. Avoiding overprovisioning and resource wastage is essential in a cloud environment. TSDACS achieves these goals through its combined approach. Experimental results demonstrate that TSDACS consistently outperforms existing deadline-aware scheduling algorithms such as CPDALB, DBS, and RDLBS2 across key metrics, including deadline miss ratio, makespan, response time, and monetary gains. It achieves a notably low deadline miss ratio of 1.22% and records the lowest makespan and response time, with improvements of at least 2.16% and 2.45%, respectively, indicating faster and more efficient cloudlet execution. Financially, TSDACS provides the highest total gain and the lowest total loss, achieving up to 75% more gain and 90% less loss than the other comparison algorithms. While CPDALB leads slightly in VM utilization and load balancing, TSDACS proves more effective in meeting deadlines by maintaining better VM utilization and load distribution, which is critical in time-sensitive cloud environments. TSDACS records the fewest deadline misses and the lowest miss ratio, along with improved overall performance, it confirms its reliability for real-time, deadline-driven cloud applications.

The performance of the TSDACS algorithm can be further enhanced by implementing AI-driven auto-scaling, which adjusts VM capacity in real time based on workload variations, thereby improving the performance efficiency. Additionally, TSDACS can be extended to support energy-aware scheduling, minimizing energy consumption while still meeting deadlines, ultimately reducing operational costs.

Conflict of Interest: The authors declare no conflicts of interest(s).

Data Availability Statement: The Datasets used and /or analysed during the current study available from the corresponding author on reasonable request.

Funding: No fundings.

Consent to Publish: All authors gave permission to consent to publish.

References

[1] Alkaam, Nora Omran, et al. "Hybrid Henry Gas-Harris Hawks Comprehensive-Opposition Algorithm for Task Scheduling in Cloud Computing." IEEE Access (2025).

[2] Goubaa, Aicha, et al. "Scheduling periodic and aperiodic tasks with time, energy harvesting and precedence constraints on multi-core systems."Information Sciences 520 (2020): 86-104.

[3] Asiaban, Sedigheh, Mohsen Ebrahimi Moghaddam, and M. Abbaspour. "A Real-Time Scheduling Algorithm for Soft Periodic Tasks."International Journal of Digital Content Technology and its Applications 3.4 (2009.

[4] Visheratin, Alexander, et al. "Hard-deadline constrained workflows scheduling using metaheuristic algorithms."Procedia Computer Science 66 (2015): 506-514.
[5] Singh, Jagbeer. "An algorithm to reduce the time complexity of earliest deadline first scheduling algorithm in real-time system." arXiv preprint arXiv:1101.0056 (2010).
[6] Naghibzadeh, Mahmoud. "A modified version of rate-monotonic scheduling algorithm and its' efficiency assessment."Proceedings of the Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable Systems. (WORDS 2002). IEEE, 2002.
[7] Zhang, Wei, et al. "An improved least-laxity-first scheduling algorithm of variable time slice for periodic tasks."6th IEEE International Conference on Cognitive Informatics. IEEE, 2007.
[8] Shinde, Vijayshree, and Seema C. Biday. "Comparison of real time task scheduling algorithms."Int. J. Comput. Appl 158.6 (2017): 37-41.
[9] Scordino, Claudio, and Giuseppe Lipari. "A resource reservation algorithm for power-aware scheduling of periodic and aperiodic real-time tasks." IEEE Transactions on Computers 55.12 (2006): 1509-1522.
[10] Yao, Fuguang, Changjiu Pu, and Zongyin Zhang. "Task duplication-based scheduling algorithm for budget-constrained workflows in cloud computing." IEEE Access 9 (2021): 37262-37272.
[11] Yu, Lei, Fei Teng, and Frederic Magoules. "Node scaling analysis for power-aware real-time tasks scheduling." IEEE Transactions on Computers 65.8 (2015): 2510-2523.
[12] Khan, Ayaz Ali, et al. "A migration aware scheduling technique for real-time aperiodic tasks over multiprocessor systems." IEEE Access 7 (2019): 2785-2773.
[13] Lu, Lei, et al. "Application-driven dynamic vertical scaling of virtual machines in resource pools." 2014 IEEE Network Operations and Management Symposium (NOMS). IEEE, 2014.
[14] Sotiriadis, Stelios, et al. "Vertical and horizontal elasticity for dynamic virtual machine reconfiguration." IEEE Transactions on Services Computing 99 (2016): 1-1.
[15] Alyas, Tahir, et al. "Performance Framework for Virtual Machine Migration in Cloud Computing." Computers, Materials & Continua 74.3 (2023).
[16] Shahapure, Nagamani H., and P. Jayarekha. "Distance and traffic based virtual machine migration for scalability in cloud computing." Procedia computer science 132 (2018): 728-737.
[17] Beitollahi, Hakem, Seyed Ghassem Miremadi, and Geert Deconinck. "Fault-tolerant earliest-deadline-first scheduling algorithm." 2007 IEEE International Parallel and Distributed Processing Symposium. IEEE, 2007.
[18] Li, Qi, and Wei Ba. "A group priority earliest deadline first scheduling algorithm." Frontiers of Computer Science 6 (2012): 560-567.
[19] Tseng, Li-Y., Yeh-H. Chin, and Shu-Ching Wang. "A deadline-based task scheduling with minimized makespan." International Journal of Innovative Computing, Information and Control 5.7 (2009): 1665-1679.
[20] Rodriguez, Maria Alejandra, and Rajkumar Beya. "Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds." IEEE transactions on cloud computing 2.2 (2014): 222-235.
[21] Sidhu, Harmanbir Singh. "Cost-deadline based task scheduling in cloud computing." 2015 Second International Conference on Advances in Computing and Communication Engineering. IEEE, 2015.
[22] Nayak, Suvendu Chandan, and Chitaranjan Tripathy. "Deadline based task scheduling using multi-criteria decision-making in cloud environment." Ain Shams Engineering Journal 9.4 (2018): 3315-3324.
[23] Ben Alla, Said, et al. "An efficient energy-aware tasks scheduling with deadline-constrained in cloud computing." Computers 8.2 (2019): 46.
[24] Li, Jianpeng, et al. "Task scheduling algorithm for heterogeneous real-time systems based on deadline constraints." 2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC). IEEE, 2019.

[25] Sahoo, Sampa, Bibhudatta Sahoo, and Ashok Kumar Turuk. "A learning automata-based scheduling for deadline sensitive task in the cloud." IEEE Transactions on Services Computing 14.6 (2019): 1662-1674.
[26] Tarafdar, Anurina, et al. "Energy and makespan aware scheduling of deadline sensitive tasks in the cloud environment." Journal of Grid Computing 19 (2021): 1-25.
[27] Zhang, Yu, et al. "Deadline-aware dynamic task scheduling in edge–cloud collaborative computing." Electronics 11.15 (2022): 2464.
[28] He, Xiaojian, et al. "A two-stage scheduling method for deadline-constrained task in cloud computing." Cluster Computing 25.5 (2022): 3265-3281.
[29] Iranmanesh, Amir, and Hamid Reza Naji. "DCHG-TS: a deadline-constrained and cost-effective hybrid genetic algorithm for scientific workflow scheduling in cloud computing." Cluster Computing 24 (2021): 667-681.
[30] Azizi, Sadoon, et al. "Deadline-aware and energy-efficient IoT task scheduling in fog computing systems: A semi-greedy approach." Journal of network and computer applications 201 (2022): 103333.
[31] Verma, Amandeep, and Sakshi Kaushal. "Deadline constraint heuristic-based genetic algorithm for workflow scheduling in cloud." International Journal of Grid and Utility Computing 5.2 (2014): 96-106.
[32] Komarasamy, Dinesh, and Vijayalakshmi Muthuswamy. "Adaptive deadline based dependent job scheduling algorithm in cloud computing." 2015 Seventh International Conference on Advanced Computing (ICoAC). IEEE, 2015.
[33] Ohee, Muhammad Makama Mahmudur Rahman, et al. "An Efficient Deadline Based Priority Job Scheduling in Mobile Cloud Computing." IET Communications 19.1 (2025): e70031.
[34] Abdi, Somayeh, Mohammad Ashjaei, and Saad Mubeen. "Deadline-constrained security-aware workflow scheduling in hybrid cloud architecture." Future Generation Computer Systems 162 (2025): 107466.
[35] Qamar, Saad, Nesar Ahmad, and Parvez Mahmood Khan. "Task Scheduling for Public Clouds Using a Fuzzy Controller-Based Priority-and Deadline-Aware Approach." Future Internet 17.4 (2025): 148.
[36] Effah, Emmanuel, et al. "Exploring the Landscape of CPU Scheduling Algorithms: A Comprehensive Survey and Novel Adaptive Deadline-Based Approach." International Journal of Computer Science and Information Security (IJCSIS) 23.1 (2025).
[37] Ma, Xiaojin, et al. "An IoT-based task scheduling optimization scheme considering the deadline and cost-aware scientific workflow for cloud computing." EURASIP Journal on Wireless Communications and Networking 2019.1 (2019): 1-19.
[38] Anwar, Nazia, and Huifang Deng. "Elastic scheduling of scientific workflows under deadline constraints in cloud computing environments." Future Internet 10.1 (2018): 5.
[39] Haidri, Raza Abbas, Chittaranjan Padmanabh Katti, and Prem Chandra Saxena. "Capacity based deadline aware dynamic load balancing (CPDALB) model in cloud computing environment." International Journal of Computers and Applications 43.10 (2021): 987-1001.
[40] Alworafi, Mokhtar A., and Suresha Mallappa. "A collaboration of deadline and budget constraints for task scheduling in cloud computing." Cluster Computing 23.2 (2020): 1073-1083.
[41] Haidri, Raza A., et al. "A deadline aware load balancing strategy for cloud computing." Concurrency and Computation: Practice and Experience 34.1 (2022): e6496.