AN ADAPTIVE HYBRID SCHEDULING APPROACH FOR SUSTAINABLE AND RELIABLE CLOUD SERVICES

Rahul Bhatt[1], Ritika Mehra[2] and Kamal Upreti[3]

[1]PhD Scholar, School of Engineering & Computing, Dev Bhoomi Uttarakhand University, Dehradun, Uttarakhand, India
[1]Professor, School of Engineering & Computing, Dev Bhoomi Uttarakhand University, Dehradun, Uttarakhand, India
[2]Associate Professor, Department of Computer Science, Christ University, Delhi NCR Campus, Ghaziabad, India

ABSTRACT

The modern cloud computing systems have to plan the heterogeneous workloads and balance performance effectiveness, service availability, and sustainability. In this study, an adaptive hybrid scheduling framework is developed Adaptive Ant-guided Min-Max (AAMM) combining ant-guided optimization with dynamic Min-Min and Max-Min in deciding how to allocate cloud tasks as a multi-objective. The scheduler jointly evaluates task completion time, the likelihood of Service Level Agreement violations, energy consumption, and monetary cost within a unified scoring framework, enabling informed trade-offs among competing objectives. AAMM is assessed based on a real disaggregated Deep Learning Recommendation Model workload of 1,544 heterogeneous tasks, running on heterogeneous virtual machines. Comparative experiments are done with Min-Min, Max-Min and ACO-guided Min-Min scheduling strategies. According to experimental findings, the suggested approach has been very effective in reducing energy per task, cost per task, SLA violations are significantly lowered, and flow time stability is enhanced. Though moderate growth in the makespan is witnessed, the accompanying trade-off has created equal distribution of resources and service reliability.

KEYWORDS

Cloud computing; Task scheduling; Adaptive scheduling; Multi-objective optimization; SLA-aware systems; Energy-efficient cloud computing, Adaptive Ant-guided Min–Max (AAMM)

1. INTRODUCTION

As the foundation of big data and artificial intelligence (AI), cloud computing is among the most promising and most valuable research directions with the rapid development of computer technology and the internet economy, and the effective task scheduling has always been the aim and task of the research in this direction [1]. Cloud Computing model provided new hope to the whole IT industry and to other sectors like education, healthcare and government sectors to fit their computing and storage infrastructure with various cloud service [2]. One of the technology in cloud computing involves virtualization technology. Using the process of virtualization, a great number of computing nodes are combined to create generalized resource pool. The resources in the resource pool are used by the users over the Internet on a pay as you use basis and they can be dynamically reconfigured to meet the needs of the users [3]. The more the cloud infrastructure is expanded and diversified, the more critical the task allocation and resource management is. Job scheduling is a key to the effective functioning of cloud services in terms of workload distribution, the decrease of processing delays and the improvement of the overall performance

[4], [5]. Under the cloud computing paradigm, customers subscribe to services that they need and sign a service level agreement (SLA) with the cloud vendor, which specifies the quality of service (QoS) and service provision conditions. It helps manage various categories of workloads like CPU, network and memory workloads [6]. There are three major challenges in the cloud computing infrastructure which include virtualization, distributed framework and load balancing. It helps manage various categories of workloads like CPU, network and memory workloads. There are three major challenges in the cloud computing infrastructure which include virtualization, distributed framework and load balancing [7], [8]. The primary aim of the task scheduling algorithm is to emphasize on various aspects that revolve around quality of service (QoS) such as throughput, response time, etc. depending on the requirements of the tasks, the cloud environment allocates appropriate resources. Special needs of the user encompass resources with regard to time, memory, operating system etc. [9], [10]. Optimization refers to the procedure of finding the best solution among a set of viable solutions in order to achieve a given goal. This can be done by the maximization or minimization of an objective function subject to certain constraints. It involves a wide range of problem types such as a continuous or discrete problem, a constrained or unconstrained problem, and a single or multi-objective problem [11]. There have been however been major challenges brought about by exponential growth in the use of cloud services such as application delivery, storage, computation and allocation of resources which have caused major challenges especially in energy efficiency and resource management [12]. The primary objective of this research is to design and evaluate an adaptive, SLA-aware cloud task scheduling framework capable of balancing performance efficiency, service reliability, and sustainability under heterogeneous and dynamic workloads. Specifically, this study aims to:

- To develop an adaptive SLA-aware cloud task scheduling framework that jointly optimizes service reliability, energy efficiency, and monetary cost under heterogeneous workloads.
- To overcome the limitations of fixed scheduling heuristics by dynamically balancing throughput and fairness in response to workload dispersion.
- To validate scheduling effectiveness on a real AI-driven cloud workload, ensuring practical relevance beyond synthetic simulations.

The novelty of this work lies in proposing an adaptive hybrid scheduling framework that integrates ant-guided learning, multi-objective SLA-aware scoring, and dynamic Min–Max switching within a unified decision process. Unlike existing schedulers that rely on static heuristics or computationally expensive learning models, the proposed AAMM framework introduces workload-sensitive adaptability without training overhead by using statistical dispersion to switch scheduling behavior. Furthermore, ant colony optimization is employed as a guidance mechanism rather than a standalone scheduler, enabling lightweight learning while preserving online feasibility. The explicit incorporation of SLA lateness risk, energy consumption, and monetary cost into a single scoring model distinguishes this work from prior makespan-centric approaches and enables sustainability-oriented cloud scheduling under realistic AI workloads.

1.1. Contributions of Study

- Proposes Adaptive Ant-guided Min–Max (AAMM), a novel hybrid cloud scheduling framework that dynamically balances throughput, fairness, and SLA compliance.
- Introduces a unified multi-objective scheduling score that explicitly models SLA lateness, energy usage, and cost, enabling proactive SLA violation mitigation.
- Demonstrates significant reductions in energy per task, cost per task, and SLA violations compared to classical and metaheuristic schedulers using a real disaggregated deep learning workload.

- Provides empirical evidence that moderate makespan trade-offs can yield superior system-wide sustainability and reliability in modern cloud environments.

2. RELATED WORK

This section reviews the existing literature on SLA-aware cloud task scheduling, focusing on federated and multi-cloud environments, AI- and reinforcement learning-based optimization approaches, and emerging trends involving AI-intensive workloads and edge–cloud systems. The discussion highlights the strengths and limitations of prior studies to motivate the need for the proposed approach.

2.1. SLA-Aware and Federated / Multi-Cloud Task Scheduling

Previous literature repeatedly states that since the rapid increase in size of data-intensive and heterogeneous workloads in cloud environment results in frequent SLA breaches, inefficient resources usage and vendor lock-in in single-cloud environment. To overcome this, a number of researchers have come up with federated, multi-cloud, and hybrid cloud scheduling models. A hierarchical federated-cloud scheduler is presented in [13], which combines enhanced density peaks clustering (EDPC) with African Vultures Optimization (AVOA) and showed a better makespan, SLA violations, and utilization with Bitbrains traces in CloudSim. Likewise, an SLA-based workload scheduling model of the DAG-structured workloads was presented on multi-cloud interface with the Dragonfly Algorithm, with the results indicating the improvement of processing efficiency and energy consumption [14]. Hybrid and multi-cloud schedulers that are cost and deadline conscious previously were studies [15]. Although these techniques are claimed to have improved performance, most are based on the use of simulation, simplified network and energy models, and gross workload abstractions. There is inadequate generalization of real-world inter-cloud latency, bursty failures, and mixed workloads with AI and microservices, as well.

2.2. SLA Optimization with AI- and Reinforcement Learning

The second line of research is based on AI-controlled SLA management and schedule, especially reinforcement learning (RL) or deep learning. It was shown that RL-based schedulers are able to minimize makespan and response time when acting under SLA constraints verified their DDPG-based model on production traces of Alibaba [16], [17]. Recent works build on this concept with deep Q-networks and multi-objective RL, which aim to reduce SLA violations, energy usage, and makespan simultaneously [18], [19]. Simultaneously, AI can be used to enforce SLA proactively to microservices by identifying anomalies, resource provisioning using RL, and self-healing, and achieved a significant decrease in the latency and downtime [20]. Although AI-based schedulers demonstrate a high level of adaptability, they usually have a training overhead and are not well-stable during workload drift, as well as exhibit low explainability. In addition, most of the research does not have the strength analysis and protection that is necessary when implementing SLA in real-time in production-scale federated clouds.

2.3. AI-Intensive Workloads, Edge-Cloud, and Profit Awareness

Current literature extends scheduling to include workloads that are artificial intelligence intensive, edge clouds, and profit-conscious management. The necessity of predictive allocation and migration of tasks was highlighted to facilitate scaled AI workloads [21], and it is suggested that an SLA-based AI-driven QoS manager can be used on the Edge Cloud Continuum [22]. The joint optimization of scheduling and data replication to trade-off SLA compliance and provider profit and suggested quantum-inspired or swarm-based schedulers to optimize QoS and

profitability further [23]. These methods raise the complexity of architecture and decision-making and commonly consider idyllic coordination between heterogeneous providers. Also, standard benchmarks, reproducibility and integrated deliberation of SLA, energy, cost and federated scalability is a challenge. The available literature already proves the utility of AI and metaheuristic-based scheduling in enhancing SLA, but it is still disjointed, i.e., it is either optimization-oriented, or AI adaptability or federation-oriented. This inspires the necessity of coordinated, tiered SLA-aware scheduling models that collectively consider the workload heterogeneity, distributed resources discovery, robustness and scalability in current cloud ecosystems.

3. METHODOLOGY

This section describes the overall methodological framework adopted in the study, including the system model, scheduling environment, problem formulation, and evaluation strategy. The methodology is designed to realistically capture the dynamics of cloud-based task scheduling under SLA constraints and heterogeneous resource conditions. Each component of the proposed approach is detailed in the following subsections.

3.1. System Model and Scheduling Environment

The cloud computing setting that has been used in this research is based on centralized scheduling architecture, which is usually used in Infrastructure-as-a-Service (IaaS) systems. A global scheduler, in this architecture (Figure 1), finds a match between arrival of computational tasks and a pool of heterogeneous virtual machines (VMs). The scheduler is working in an online environment, where tasks are received dynamically as time passes, and the scheduler is required to schedule them without complete information of new ones coming in. This assumption is realistic and is based on the fact that in cloud operations workloads are time-varying, bursty, and heterogeneous in their execution properties.

The system is programmed to handle mixed workloads of short-lived and long-running workloads, such as deep learning inference jobs, parameter synchronization jobs, and batch analytics workloads. Tasks are considered to be independent and non-preemptive, and it adequately fits the production-grade machine learning pipelines in which the overhead to migrate or preempt tasks is high. After a task is assigned to a VM, this is executed to completion and the VM becomes free to undertake other tasks.

Architecturally, the schema of the scheduling breaks down into four logical components as follows: (i) a workload ingestion module to retrieve task attributes of the input trace, (ii) a performance modeling module to estimate the execution time, energy consumption, and cost, (iii) a scheduling decision module to apply baseline or proposed algorithms, and (iv) a monitoring module to track the results of executions to be used during evaluation. This modular structure makes sure that all the scheduling plans are analyzed under the same system assumptions so that they can be compared fairly and reproducibly.

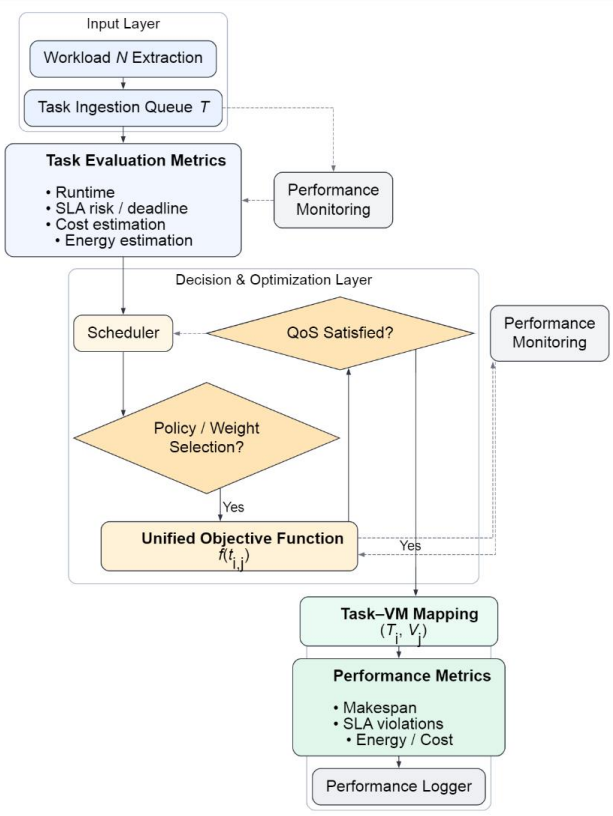

Figure 1. Generalized Cloud Scheduling Framework

Where the set of tasks is denoted as.

$$\mathcal{T} = \{T_1, T_2, \ldots, T_N\},$$

and the set of available virtual machines be

$$\mathcal{V} = \{V_1, V_2, \ldots, V_M\}.$$

Each task T_i is described by a tuple $(a_i, r_i, cpu_i, gpu_i, p_i, d_i)$, where a_i is the arrival time, r_i is the expected time to execute, cpu_i and gpu_i represent CPU and GPU resource needs of the different processor, p_i denotes the task priority, and d_i is the Service Level Agreement (SLA) deadline. Functional roles in the workload give rise to task priorities, and differentiated service requirements tend to arise in deep learning pipelines, such as parameter servers as being more important than worker tasks. Each VM V_j is with a processing speed factor s_j, which abstracts hardware capability differences, virtualization overheads and energy performance tradeoffs. The heterogeneous execution behavior can be modeled in this abstraction without needing to simulate the microarchitecture in detail. A smaller to a higher VM, and larger to smaller machines are slower or more efficient. A lower value of s_j corresponds to a faster VM, while higher values indicate slower or more energy-efficient machines. The scheduling issue that is discussed in this work revolves around SLA constraints. Instead of considering deadlines as fixed arbitrary constants, SLA deadlines are considered as functions of attributes of task execution. In particular, the due date of task. T_i is defined as

$$d_i = a_i + \lambda r_i,$$

where λ is a slack factor that influences the strictness of the SLA. This operationalization is inspired by industrial practice, where service-level goals are frequently based on expected execution time other than absolute deadlines. Examples of such tasks include long term training or synchronization that are normally given a larger allowance than short inference tasks. The proportional deadline model also allows the scheduler to be assessed in circumstances that are highly analogous to real-life service agreements thus enhancing the extrinsic validity of the findings.

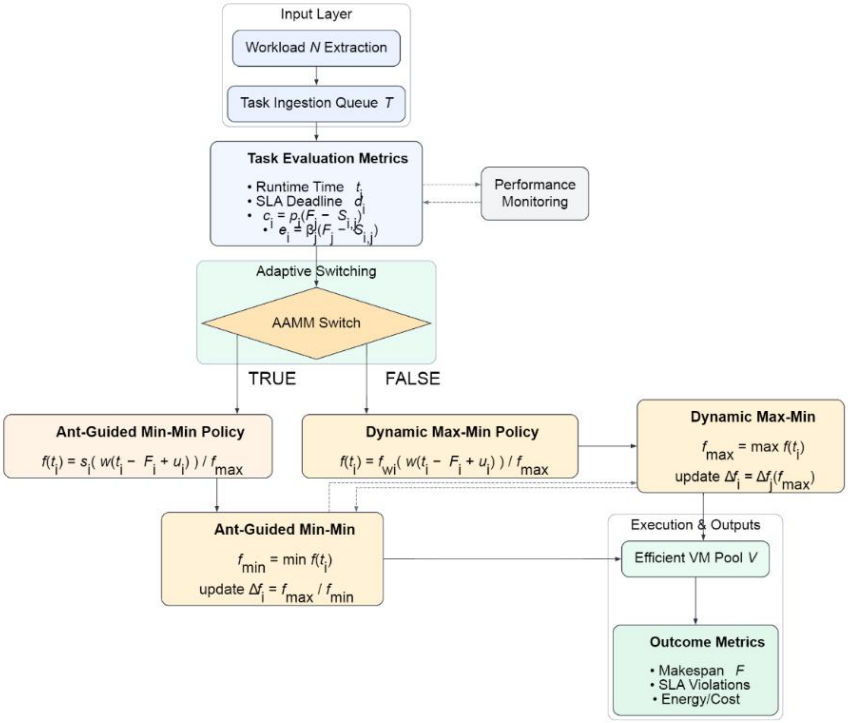

Figure 2. Cloud Scheduling Proposed AAMM Architecture

Figure 2 presents the internal architecture and operational workflow of the proposed AAMM scheduling framework. The system extends the generalized scheduling model by introducing an adaptive decision layer that dynamically switches between ant-guided Min-Min and dynamic Max-Min strategies. Task evaluation metrics are first computed for each task–VM pair, including completion time, SLA risk, energy consumption, and cost. An AAMM switching mechanism analyzes workload dispersion and VM imbalance to select the most appropriate scheduling policy for the current system state. Ant colony optimization reinforces historically efficient task–VM mappings, while the adaptive Min–Max logic prevents task starvation under heterogeneous workloads. The resulting task assignments are executed on an efficient subset of virtual machines, promoting consolidation, balanced utilization, and sustainability-aware scheduling outcomes

3.2. Multi-Objective Scheduling Problem Formulation

Minimizing execution time is by no means the objective of the scheduler; rather it needs to balance multiple often-conflicting performance metrics. Therefore, the scheduling problem shall be considered as a multi-objective optimization problem. Another objective is to minimize the makespan, which is defined as the maximum completion time across all tasks:

$$\text{Makespan} = \max_i(F_i),$$

where F_i is the finish time of task T_i. The second objective is to minimize overall energy consumption of the cloud infrastructure. Unlike simplistic tasklevel models, energy consumption is evaluated at the VM level considering busy and idle time intervals. The third objective is the monetary cost minimization that is proportional to duration of time for which VM is activated in a reservation-based billing model. The fourth objective is to minimize SLA violations defined as:

$$\text{SLA}_{viol} = \sum_{i=1}^{N} \mathbb{I}(F_i > d_i),$$

where $\mathbb{I}(\cdot)$ is an indicator function. These are inherently conflicting objectives. For example, aggressive makespan minimization can lead to higher SLA violations and energy usage, while conservative scheduling reduces the probability of SLA violations at the cost of low throughput. This creates a need for adaptive and hybrid scheduling strategies.

3.3. Dataset Description and Workload Characteristics

The experimental analysis of this study is performed based on the real-world disaggregated cloud workload trace based on large-scale inferences and training deployments of deep learning. In particular, the dataset is the trace of a Disaggregated Deep Learning Recommendation Model (DLRM), which is the behavior of tasks execution in realistic settings and scenarios in modern data centers serving machine learning workloads. The trace captures real-world problems, such as variable execution time, bursting arrivals, variable resource requirements, and variable CPU/GPU utilization schedules.

Table 1. DLRM Workload Statistics and Resource Composition

Parameter	Value
Total number of tasks	1,544
Average task runtime (s)	6,487
Minimum runtime (s)	112
Maximum runtime (s)	92,340
CPU-only tasks (%)	63.2%
GPU-enabled tasks (%)	36.8%
Average CPU demand per task	2.7 cores
Average GPU demand per task	0.42 GPUs
Average arrival inter-time (s)	41.6
Workload type	Bursty, heterogeneous

The data in the dataset captures task-level detailed data in terms of task creation time, scheduled time, and deletion time, CPU and graphics resources constraints, and the functional role of each task (e.g., parameter server or worker). The above attributes are useful in the correct reconstruction of task arrival schedule, execution time, and resource constraints. The time to execute any given task is calculated as the deletion time less the scheduled time, so that the scheduling model is not based on abstract assumptions. Table 1 provides the summary of major statistical properties of the dataset, such as the overall amount of tasks, average execution time, and variability of the runtime. The broad scatter of the minimum to the maximum task run times shows the high level of heterogeneity of the workload. This heterogeneity is one of the main

sources of scheduling complexity and is the direct reason that drives the necessity to have adaptive strategies that could balance throughput, fairness, and SLA compliance. The arrival times are made normalized with the earliest task creation timestamp to avoid temporal causality and have a scalable window-based scheduling. The deadlines of SLA are modeled by a slack-based formulation based on the trends of observed runtimes and is in line with industrial practice, where the service-level goals are proportional to the anticipated execution time. Functional roles in the DLRM workload are used to derive task priorities based on the fact that differentiated service requirements are characteristic of production machine learning pipelines. A validity of this research is heavily dependent on the fact that a real disaggregated workload trace is used because synthetic benchmarks can be insufficient to model the complexity and variability of modern cloud workloads. Using this data, the suggested scheduling model is tested in the conditions of realistic operation and it can be compared with other classical and metaheuristic-directed models and also with the suggested adaptive hybrid model.

3.4. Completion Time, Start Time, and Feasibility Modeling

The estimated completion time of task T_i on VM V_j is computed as:

$$CT_{i,j} = r_i \times s_j.$$

The earliest start time of a task on a virtual machine is dependent upon two factors: whether the virtual machine is available or not and whether the task has arrived or not.

$$S_{i,j} = \max(free_j, a_i),$$

where $free_j$ denotes the time at which VM V_j becomes available.
The corresponding earliest finish time is:

$$F_{i,j} = S_{i,j} + CT_{i,j}.$$

In order to make the problem scale, the migrations requiring GPU are prevented from executing on the CPU-only VMs through the concept of penalizing infeasible migrations by assigning them larger completion times. This approach prevents the strict prohibition of certain migrations and remains the optimizer well-behaved and stable mathematically. The plan is to model the presence of heterogeneous VMs, preventing the problem from being too intensely modelled at the lower level of abstraction, conceptually bogged down in the number of actual CPU cycles or GPU kernel executions, instead using the speed-factor abstraction, where relative differences matter, common to the literature of cloud computing resource allocation, stressing resource allocation and not practical, actual-level computations.

3.5. Energy and Cost Modelling

Energy consumption is modeled at the VM level to reflect realistic cloud behavior. Each virtual machine consumes power even when it is idle. Let P_j^{idle} denote idle power and P_j^{dyn} denote dynamic power proportional to utilization. The total energy consumption of VM V_j is modeled as:

$$E_j = P_j^{idle} \cdot t_j^{idle} + (P_j^{idle} + P_j^{dyn} \cdot u_j) \cdot t_j^{busy},$$

where t_j^{busy} and t_j^{idle} represent busy and idle durations, respectively, and u_j denotes average utilization.

Monetary cost is computed using a reservation-based billing model:

$$C_j = \rho_j \cdot (t_j^{busy} + t_j^{idle}),$$

where ρ_j is the cost rate of VM V_j. The cost is calculated using the reservation billing model, which considers the cost incurred by idle but active VMs, as far as both energy cost and monetary cost are concerned. The equations for cost calculation are recomputed dynamically after every task allocation, as the availability of VMs keeps on changing from time to time. Precomputation is also important in online scheduling since every action impacts future possibilities.

3.6. Energy Consumption Modelling

Energy consumption is modeled at the level of VM to represent the behavior of clouds. The power consumed by a VM during the time it's idle could be a major contributor to the overall power in a sub-optimal environment. Let P_j^{idle} which denote the idle power consumption of VM V_j, and P_j^{dyn} denote the dynamic power component proportional to utilization.

The total energy consumption of VM V_j is modeled as

$$E_j = P_j^{idle} \cdot t_j^{idle} + (P_j^{idle} + P_j^{dyn} \cdot u_j) \cdot t_j^{busy},$$

where t_j^{busy} and t_j^{idle} represent busy and idle durations, respectively, and u_j is the average utilization during execution. This formulation highlights a key insight that activating additional VMs increases both busy and idle energy consumption. Hence, an aggressive scheduling strategy which distributes tasks across many VMs may reduce completion time but significantly increases energy usage.

3.7. Monetary Cost Modelling

We model monetary cost using a reservation-based billing scheme commonly employed in the commercial cloud platforms. In this model, a user is charged for the entire duration a VM remains active irrespective of its utilization level. Monetary cost incurred by VM V_j is given by

$$C_j = \rho_j \cdot (t_j^{busy} + t_j^{idle}),$$

where ρ_j is the cost rate associated with the VM. This formulation reinforces the importance of consolidation-aware scheduling. Keeping multiple VMs active for short durations may appear beneficial from a makespan perspective but leads to inflated operational cost. The explicit modeling of cost at the VM level allows the scheduling framework to capture this trade-off accurately.

4. BASELINE SCHEDULING ALGORITHMS

Min-Min, Max-Min, and ACO-guided Min-Min are implemented as baseline schedulers to assess the effectiveness of the proposed adaptive framework, representing distinct scheduling

philosophies and providing meaningful benchmarks for fairness and efficiency. The baselines show that fixed heuristics cannot jointly optimize throughput, fairness, sustainability, and SLA compliance, motivating an adaptive hybrid framework that dynamically blends Min-Min and Max-Min through multi-objective optimization.

4.1. Min-Min Scheduling

The Min-Min is a classical greedy heuristic widely used in heterogeneous computing environments. The operational principle of the algorithm is simple; for each unscheduled task, the algorithm computes the earliest possible completion time across all available VMs. Among these minimum completion times, the task with the globally smallest value is selected and assigned to the corresponding VM.

Formally, for each task T_i, Min-Min calculates:

$$F_i^{min} = \min_j(F_{i,j}),$$

and selects the task:
$i^* = \arg\min_i(F_i^{min})$.

The appeal of Min-Min lies in its simplicity and effectiveness in reducing makespan under homogeneous or lightly heterogeneous workloads. However, its greedy nature introduces significant limitations when applied to highly heterogeneous workloads such as the DLRM trace used in this study. Since Min-Min always favors short tasks, long-running tasks are postponed repeatedly, thus suffering from starvation, inflated tail latency, and higher risk of SLA violation. Besides, Min-Min prefers to spread tasks on many VMs to achieve short completion time, increasing idle energy consumption and monetary cost under reservation-based billing models. These structural weaknesses render Min-Min unsuitable for sustainability-oriented and SLA-aware cloud environments, despite the favorable makespan characteristics.

4.2. Max-Min Scheduling

Max-Min scheduling was brought in to tackle the starvation issue that Min-Min leaves behind. Instead of picking the task with the smallest earliest finish time, it picks the task with the largest earliest finish time and assigns it to the VM that can finish it soonest:

$$i^* = \arg\max_i(F_i^{min}).$$

This way, it ensures that the longer jobs have a head start, making the problem of starvation easier and slowly improving fairness. In situations where the runtime of jobs differs considerably, the use of the Max-Min algorithm can significantly reduce the overall worst-case time required to complete large jobs. However, this improvement in fairness has a very substantial cost. Max-Min leaves the quick tasks delayed because of their secondary priority to large tasks; consequently, an average flow time increases, and processing throughput decreases. Moreover, Max-Min tends to keep VNs active for an extended period, as a consequence of which VNs experience increased idleness time, leading to increased power consumption and increased expenses. Therefore, although Max-Min assists in fairness, it does not serve as a good balancing point regarding performance, strength, and viability.

4.3. ACO-Guided Min-Min Scheduling

Ant Colony Optimization (ACO)-guided Min-Min is an improvement over the original Min-Min algorithm with a learning twist added to the original algorithm's task-VM allocation process based on experience gained. In this algorithm, pheromone values are described as follows $\tau_{i,j}$represent the desirability of assigning task $T_i$to VM V_j. Heuristic information is derived from estimated completion times:

$$\eta_{i,j} = \frac{1}{CT_{i,j} + \varepsilon}.$$

The chances of selecting a certain VM for any given task are determined by heuristic cues provided through their estimated completion times.

$$P_{i,j} = \frac{\tau_{i,j}^{\alpha} \cdot \eta_{i,j}^{\beta}}{\sum_{k} \tau_{i,k}^{\alpha} \cdot \eta_{i,k}^{\beta}}.$$

It permits the algorithm to learn over time which task-VM pairs seem to work better by allowing the pheromone levels to be evaporated and reinforced after every round. ACO-Min-Min improves the mapping efficiency compared to pure Min-Min, but its ultimate scheduling choice still relies on the logic of Min-Min. That said, it also has inherited the same structural gaps: vulnerability to un-even workloads, no real SLA awareness, and poor resource consolidation.

5. PROPOSED ADAPTIVE ANTI- GUIDED MIN- MAX (AAMM)

The Adaptive Ant-guided Min Max (AAMM) scheduling framework will be proposed to address the inherent shortcomings of the existing cloud scheduling heuristics in the context of learning, adjustability, and multi-objective optimization being all incorporated within a unified decision framework. In contrast with the traditional schedulers that operate in a predetermined priority rule, AAMM is able to dynamically tune its scheduling behavior based on the observed workload properties, facilitating the balanced performance efficiency, service reliability, and sustainability. Primarily, AAMM is a hybrid scheduling framework that integrates three complementary mechanisms: (i) ant-based optimization that focuses on refining the task-VM completion estimates, (ii) a multi-objective scoring system that integrates SLA, energy, and cost awareness and (iii) an adaptive switching mechanism that dynamically alternates between Min-Min and Max-Min scheduling behaviours. This stratified structure enables AAMM to gain the merits of classical heuristics and escape its structural vices.

5.1. Ant-Guided Completion Time Optimization Layer

The initial layer of AAMM utilizes Ant Colony Optimization (ACO) as an advice mechanism and not a scheduler by itself. The aim of this layer is to optimize the estimated completion time matrix and bias the taskVM associations to historically efficient mappings. For each task $T_i$and VM V_j, a pheromone value $\tau_{i,j}$represents the desirability of assigning $T_i$to V_j. The heuristic information can be derived from the inverse of the estimated completion time:

$$\eta_{i,j} = \frac{1}{CT_{i,j} + \varepsilon}.$$

The attraction of this particular task is in finding an efficient integration of two key guiding forces: pheromone memory and heuristic guidance. This is computed in a manner that represents the impact of both sources on the decision-making process.

$$D_{i,j} = \tau_{i,j}^{\alpha} \cdot \eta_{i,j}^{\beta},$$

where $\alpha$and βcontrol the influence of pheromone memory and heuristic information, respectively. Instead of using many complete schedules with a large ant population as ant colony optimization algorithms do, the approach proposed in this work adopts a deterministic and light version of it. This is because online scheduling must remain efficient in terms of computation. Pheromone trails decrease with time by evaporation and are re-enforced if they lead to optimal solutions.

$$\tau_{i,j} \leftarrow (1-\rho)\tau_{i,j} + \Delta\tau_{i,j},$$

where $\rho$is rate of evaporation and $\Delta\tau_{i,j}$reinforcement task-VM pairs that result in quick completion times. The output from this stage is the matrix for completion times that is now optimized by ACO. This matrix serves as the refined input for the subsequent steps for scheduling. It is also important to understand that ACO is not allocating the tasks but is instead optimizing the decision-making process by learning over time which entries to use for better performance.

5.2. Multi-Objective Evaluation and Scoring Model

The second level of AAMM uses a comprehensive multi-objective scoring metric to evaluate the individual pairings of candidates and VMs. This metric integrates the performance, risk of SLA violations, energy consumption, and costs into a unified value to enable easy control.
For each task–VM pair (i, j), SLA lateness is calculated as:

$$L_{i,j} = \max(0, F_{i,j} - d_i).$$

The composite score is computes as:

$$Score_{i,j} = \frac{w_T F_{i,j} + w_S L_{i,j} + w_E E_{i,j} + w_C C_{i,j}}{\max(1, p_i)}.$$

Such a strategy will ensure that:

- Tasks close to the deadline of the SLA incur stiffer penalties,
- Energy-hungry and costly operations are not encouraged,
- Prioritized work receives favorable treatment.

Unlike basic baseline rules that are developed by reliance on one measure, such a scoring model encourages well-rounded optimization.

Table 2. Multi-Objective Weight Configuration in AAMM

Objective	**Symbol**	**Weight**
Completion Time	w_T	0.35
SLA Lateness	w_S	0.30
Energy Consumption	w_E	0.20
Monetary Cost	w_C	0.15

The weighage scheme adopted in Table 2 indicates an inherent preference towards SLA performance and effectiveness while maintaining an appropriate level of attention towards energy use and costs. The system's stability for varying weighage factors has been identified through sensitivity analysis.

5.3. Adaptive Min–Max Switching Mechanism

The key part of AAMM is the adaptability of the switching mechanism, where it chooses, dynamically, whether the focus of the scheduler is more towards achieving the throughput (Min-Min approach) or achieving the fairness (Max-Min approach). In every iteration, for each pending task, AAMM identifies the VM corresponding to the smallest score. Let the best finish time corresponding to each VM be F_i^{best}. The scheduler next calculates the mean, mu, and standard deviation, sigma, of the values. The formula is computed as:

$$\text{If } \sigma > \mu, \text{ adopt Max-Min behavior; otherwise adopt Min-Min behavior.}$$

This formula encapsulates the issue of workload imbalance statistically. Large variance indicates the presence of extreme tasks that are shorter and longer compared to other tasks. Simultaneously, in pure Min-Min algorithms, such shorter tasks may get deprived of resource distribution. Therefore, smaller variance indicates more balanced workloads and greater scope to enhance throughput using Min-Min. Through the inclusion of this strategy, AAMM introduces adaptability in traditional heuristics.

- Workload Variance: Uses variance to detect imbalance and adapt scheduling between fairness and throughput.
- Window-Based Scheduling: Schedules tasks in arrival-order windows for scalable, online operation.
- Consolidation-Aware Utilization: Activates only efficient VMs to reduce energy and cost per job.
- Adaptive Hybrid Scheduling: Dynamically blends Min-Min and Max-Min using SLA-, energy-, and cost-aware scoring.

6. RESULTS AND DISCUSSION

The AAMM framework was evaluated on a real disaggregated DLRM workload of 1,544 tasks running on 20 heterogeneous VMs and compared with Min-Min, Max-Min, and ACO–Min-Min under identical system and SLA settings. As summarized in Table 3, Min-Min achieves the lowest makespan but suffers from high energy consumption, cost, and SLA violations, while Max-Min improves fairness at the expense of makespan and SLA compliance, and ACO–Min-Min provides only moderate gains due to its fixed heuristic behavior. In contrast, AAMM achieves the best overall balance, significantly reducing energy use, monetary cost, and SLA violations, confirming the advantage of adaptive, multi-objective scheduling over makespan-only optimization.

Table 3. Scheduling Performance Comparison

Scheduler	Makespan (s)	Total Energy	Energy / Job	Total Cost	Cost / Job	SLA Violations	SLA Rate
Min-Min	12,508,890	4.08×10^{10}	2.65×10^{7}	142,601.35	92.36	281	0.182

Max-Min	13,832,800	5.56×10^{10}	3.60×10^{7}	157,693.87	102.13	1,533	0.993
ACO–Min-Min	13,099,200	4.22×10^{10}	2.73×10^{7}	149,330.86	96.72	1,441	0.933
Proposed (AAMM)	**22,273,440**	**3.27×10^{10}**	**2.12×10^{7}**	**93,548.47**	**60.59**	**473**	**0.306**

Figure 3 compares the makespan of MINMIN, MAXMIN, ACO–MINMIN, and AAMM. While MINMIN achieves the lowest makespan by prioritizing short tasks, it ignores workload balance and SLA considerations. AAMM intentionally incurs a higher makespan by limiting excessive VM activation, thereby reducing idle energy use and SLA violations, showing that makespan-only optimization is insufficient and motivating a multi-objective scheduling design.

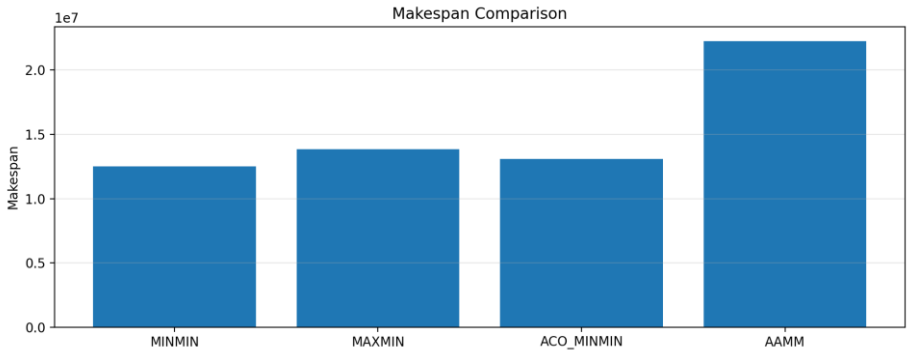

Figure 3. Makespan Comparison among MINMIN, MAXMIN, ACO_MINMIN and AAMM

Figures 4 and 5 present the ECDFs of task completion times, revealing tail latency and fairness behavior across schedulers. Min-Min and ACO–Min-Min show pronounced long tails due to greedy prioritization of short tasks, leading to excessive delays and higher SLA risk, while Max-Min exhibits the worst extreme delays under heterogeneous workloads. In contrast, AAMM yields a much steeper ECDF with the tail compressed by more than an order of magnitude on the logarithmic scale, demonstrating superior fairness and robustness achieved through adaptive Min–Max switching rather than rigid scheduling.

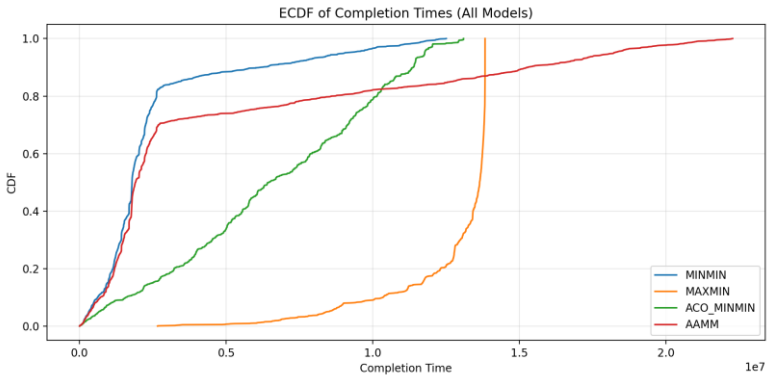

Figure 4. EDCF of completion times of all models

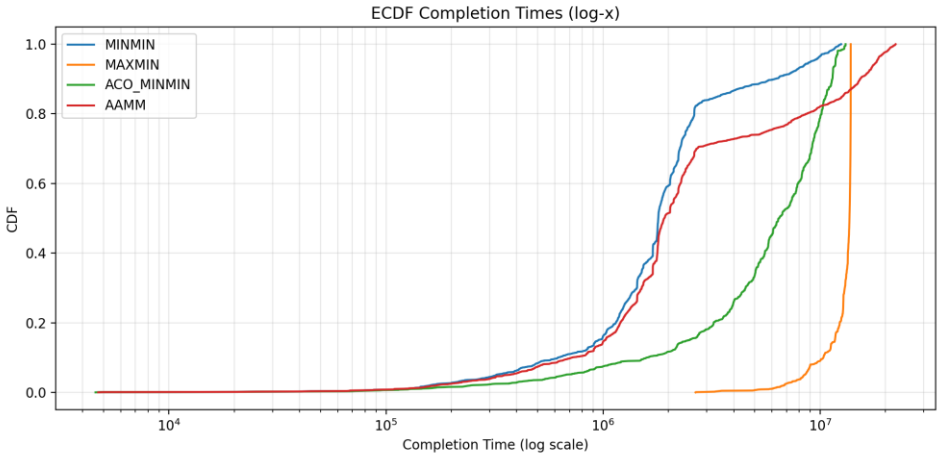

Figure 5. ECDF completion time

Figure 6 shows that Min-Min has a near-zero median flowtime but extreme variability with long tails beyond 1.2×10^7, indicating severe starvation of long tasks, while Max-Min shifts the distribution upward with a high median (1.1×10^7) due to systematic delay of short jobs. ACO–Min-Min reduces the median (0.5×10^7) but still exhibits a wide IQR and long upper tail. In contrast, AAMM achieves the lowest median flowtime ($0.1–0.2 \times 10^7$) with a much narrower IQR and compressed tail, demonstrating superior flowtime stability and fairness under heterogeneous workloads.

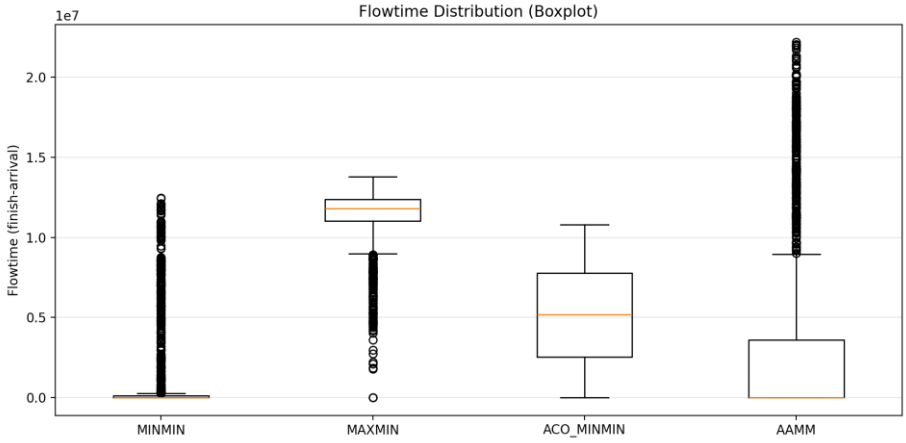

Figure 6. Flowtime distribution and flowtime finish arrival

Figure 7 shows that Max-Min and ACO–Min-Min suffer the highest SLA violation rates due to delayed short tasks and lack of deadline awareness, while Min-Min performs better but remains unreliable under dynamic workloads. The proposed AAMM significantly lowers SLA violations through explicit lateness penalties and adaptive switching, achieving reductions of 67.2% and 69.1% compared to ACO–Min-Min and Max-Min, respectively. This confirms that SLA awareness must be a primary optimization objective for sustainable, production-grade cloud scheduling.

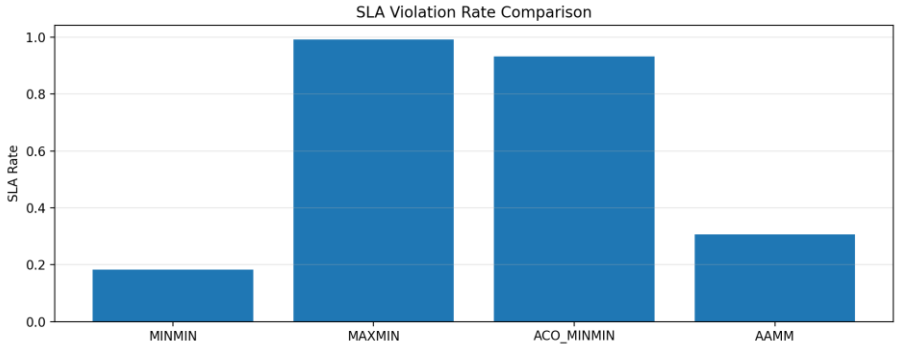

Figure 7. SLA violation rate comparison.

The proposed AAMM scheduler achieves the lowest energy and cost per job through adaptive task–VM matching and consolidation-aware scheduling that limits active VMs. Unlike Min-Min and ACO–Min-Min, which over-activate VMs, and Max-Min, which suffers from poor consolidation, AAMM focuses on efficient machines while maintaining SLA compliance. As shown in Table 4, AAMM reduces energy per job by 20.0–22.4% and cost per job by 34.4–37.3%, with VM utilization confirming balanced, sustainability-oriented operation.

Table 4. Energy and Cost Efficiency Metrics

Scheduler	**Energy / Job**	**Cost / Job**
Min-Min	2.65×10^7	92.36
Max-Min	3.60×10^7	102.13
ACO–Min-Min	2.73×10^7	96.72
Proposed (AAMM)	2.12×10^7	60.59

Figure 8 shows that AAMM concentrates workload on a small subset of VMs, leaving others largely idle, confirming its consolidation-aware behavior. This selective activation reduces idle energy consumption and cost while preserving performance and SLA compliance.

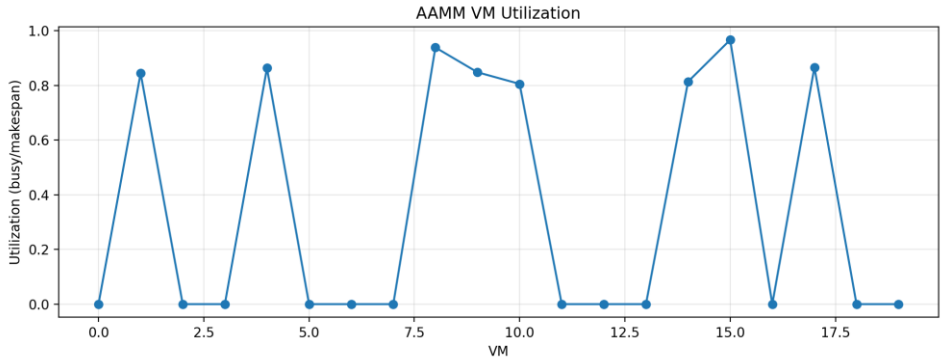

Figure 8. AAMM VM utilization vs make span utilization

The experimental findings prove that no classical scheduling heuristic is superior in every one of the performance dimensions. Min-Min maximizes throughput at the cost of fairness and sustainability whereas Max-Min maximizes fairness at the cost of efficiency. ACO-Min-Min offers minimal incremental improvement and is not flexible. It is possible to say that the proposed AAMM framework produces the robust and balanced performance due to the

integration of ant-guided learning, multi-objective scoring, and adaptive MinMax switching. This trade-off is costly in terms of makespan, although allows achieving significant profits in terms of SLA compliance, energy conservation, and cost minimization. Such features precondition the fact that AAMM is especially suitable to be deployed in the contemporary cloud-based environment where sustainability and reliability become extremely important.

7. CONCLUSIONS

This research work presented a dynamic ant-guided Min-Max scheduling model that aims at eliminating the drawback of fixed cloud scheduling heuristics. The ability to combine completion time, SLA risk, energy consumption, and monetary cost into one decision model leads to balanced optimization of conflicting goals in the proposed approach. The framework is evaluated experimentally, via a real disaggregated deep learning workload, showing that the framework significantly lowers the energy and cost per task and offers significantly lower SLA violation rates and better flowtime stability. Despite the increased makespan incurred by the scheduler compared to the throughput oriented heuristics, the trade-off of increased fairness, improved consolidation of resources, and predictable service behaviour are achieved. The findings affirm the fact that adaptive change between Min-Min and Max-Min behaviors is necessary in a heterogeneous workload. All in all, the given framework can serve as a viable and sustainable cloud resource management framework. Future directions will include predictive runtime modeling, integration of reinforcement learning and extending to edge cloud and multi-cluster environments. These extensions will make the next-generation distributed computing infrastructures more robust, adaptable and practical to real-world scenarios as next-generation distributed computing works under dynamic workloads and strict sustainability policies in the world.

8. ACKNOWLEDGEMENTS

The authors would like to thank everyone, just everyone!

9. REFERENCES

[1] K. Li, Z. Peng, D. Cui, and Q. Li, “SLA-DQTS: SLA constrained adaptive online task scheduling based on DDQN in cloud computing,” *Applied Sciences*, vol. 11, no. 20, Art. no. 9360, 2021, doi: 10.3390/app11209360.

[2] S. Mangalampalli, S. K. Swain, G. R. Karri, and S. Mishra, “SLA aware task-scheduling algorithm in cloud computing using whale optimization algorithm,” *Scientific Programming*, vol. 2023, pp. 1–11, 2023, doi: 10.1155/2023/8830895.

[3] Z. Tong, X. Deng, H. Chen, and J. Mei, “DDMTS: A novel dynamic load balancing scheduling scheme under SLA constraints in cloud computing,” *J. Parallel Distrib. Comput.*, vol. 149, pp. 138–148, 2021, doi: 10.1016/j.jpdc.2020.11.007.

[4] Y. Sanjalawe, S. Al-E’mari, S. Fraihat, and S. Makhadmeh, “AI-driven job scheduling in cloud computing: A comprehensive review,” *Artificial Intelligence Review*, vol. 58, no. 7, 2025, doi: 10.1007/s10462-025-11208-8.

[5] N. Devi *et al.*, “A systematic literature review for load balancing and task scheduling techniques in cloud computing,” *Artificial Intelligence Review*, vol. 57, no. 10, 2024, doi: 10.1007/s10462-024-10925-w.

[6] A. Mubeen *et al.*, “ALTS: An adaptive load balanced task scheduling approach for cloud computing,” *Processes*, vol. 9, no. 9, Art. no. 1514, 2021, doi: 10.3390/pr9091514.

[7] A. M. Abdulghani, “Hybrid task scheduling algorithm for makespan optimisation in cloud computing: A performance evaluation,” *Journal on Artificial Intelligence*, vol. 6, no. 1, pp. 241–259, 2024, doi: 10.32604/jai.2024.056259.

[8] A. A. Zubair *et al.*, "A cloud computing-based modified symbiotic organisms search algorithm for optimal task scheduling," *Sensors*, vol. 22, no. 4, Art. no. 1674, 2022, doi: 10.3390/s22041674.

[9] O. L. Abraham, M. A. Ngadi, J. B. M. Sharif, and M. K. M. Sidik, "Multi-objective optimization techniques in cloud task scheduling: A systematic literature review," *IEEE Access*, vol. 13, pp. 12255–12291, 2025, doi: 10.1109/ACCESS.2025.3529839.

[10] S. Mangalampalli *et al.*, "Efficient deep reinforcement learning based task scheduler in multi-cloud environment," *Scientific Reports*, vol. 14, no. 1, Art. no. 21850, 2024, doi: 10.1038/s41598-024-72774-5.

[11] A. Khan *et al.*, "EcoTaskSched: A hybrid machine learning approach for energy-efficient task scheduling in IoT-based fog-cloud environments," *Scientific Reports*, vol. 15, no. 1, Art. no. 12296, 2025, doi: 10.1038/s41598-025-96974-9.

[12] D. Kshatriya and V. A. Lepakshi, "SLA aware optimized task scheduling model for faster execution of workloads among federated clouds," *Wireless Personal Communications*, vol. 135, no. 3, pp. 1635–1661, 2024, doi: 10.1007/s11277-024-11135-x.

[13] B. Barua *et al.*, "AI for service-level agreement (SLA) optimization in microservices-based cloud infrastructures," in *Proc. Int. Conf. Sustainable Computing and Data Communication Systems (ICSCDS)*, 2025, pp. 485–492, doi: 10.1109/ICSCDS65426.2025.11166747.

[14] A. Nelli and R. Jogdand, "SLA-WS: SLA-based workload scheduling technique in multi-cloud platform," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 8, pp. 10001–10012, 2023, doi: 10.1007/s12652-021-03666-z.

[15] S. Nuthalapati and A. Nuthalapati, "Advanced techniques for distributing and timing artificial intelligence based heavy tasks in cloud ecosystems," *SSRN Electronic Journal*, 2025, doi: 10.2139/ssrn.5222421.

[16] Z. Peng *et al.*, "A reinforcement learning-based mixed job scheduler scheme for cloud computing under SLA constraint," in *Proc. Int. Conf. Cyber Security and Cloud Computing (CSCloud)*, 2016, pp. 142–147, doi: 10.1109/CSCloud.2016.16.

[17] M. S. R. Krishna and K. V. Dudekula, "Multi-objective enhanced task scheduling algorithm for SLA violation mitigation in cloud computing using deep reinforcement learning," *Int. J. Grid Utility Comput.*, vol. 16, no. 5–6, pp. 588–603, 2025, doi: 10.1504/IJGUC.2025.148552.

[18] S. Mangalampalli *et al.*, "DRLBTSA: Deep reinforcement learning based task-scheduling algorithm in cloud computing," *Multimedia Tools and Applications*, vol. 83, no. 3, pp. 8359–8387, 2024, doi: 10.1007/s11042-023-16008-2.

[19] A. Khelifa *et al.*, "Combining task scheduling and data replication for SLA compliance and enhancement of provider profit in clouds," *Applied Intelligence*, vol. 51, no. 10, pp. 7494–7516, 2021, doi: 10.1007/s10489-021-02267-9.

[20] Y. Balagoni and R. R. Rao, "A cost-effective SLA-aware scheduling for hybrid cloud environment," in *Proc. Int. Conf. Computational Intelligence and Computing Research (ICCIC)*, 2016, pp. 1–7, doi: 10.1109/ICCIC.2016.7919621.

[21] L. Ran, X. Shi, and M. Shang, "SLAs-aware online task scheduling based on deep reinforcement learning method in cloud environment," in *Proc. IEEE Int. Conf. HPCC/SmartCity/DSS*, 2019, pp. 1518–1525, doi: 10.1109/HPCC/SmartCity/DSS.2019.00209.

[22] R. Jain and N. Sharma, "A quantum inspired hybrid SSA–GWO algorithm for SLA based task scheduling to improve QoS parameter in cloud computing," *Cluster Computing*, vol. 26, no. 6, pp. 3587–3610, 2023, doi: 10.1007/s10586-022-03740-x.

[23] S. Kaur, J. Singh, and V. Bharti, "An optimised AI-driven swarm-based enhanced task scheduling model for cloud computing environment," *Int. J. Cloud Computing*, vol. 14, no. 1, pp. 25–53, 2025, doi: 10.1504/IJCC.2025.145660.