

An Integrated Qos Dependency Aware and Multi Objective Metaheuristic Framework For Cloud Task Scheduling

M.Rupasri¹, G.P.S. Varma², Hemalatha Indukuri³

¹Research Scholar, Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Guntur, Andhra Pradesh.

²Professor, Department of computer science and engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Guntur, Andhra Pradesh

³Professor, Department of IT, S.R.K.R. Engineering college, Bhimavaram Andhra Pradesh

Abstract

Cloud task scheduling involves heterogeneous virtual machines, differentiated service requirements, workflow precedence constraints, and competing time, energy, cost, and utilization objectives. We propose an integrated three-model framework. QTSF clusters independent tasks using normalized service and resource attributes, orders them by priority density, maps them to compatible virtual machines, and fixes load imbalance. DAWSM extends this framework to directed acyclic graphs through dependency-aware task clustering, a precedence-feasible discrete particle swarm representation, and constraint repair. SMMSF clusters virtual machines, normalizes four conflicting objectives, and adaptively refines schedules at scale. Formal propositions guarantee termination, unique assignment, precedence preservation, and bounded normalized fitness. QTSF achieves 5693 ms makespan, 92 percent load-balancing efficiency, and 88 percent utilization in the reported aggregate experiments; DAWSM achieves 95 percent dependency satisfaction, 4200 ms workflow time, 92 percent clustering efficiency, and 225 cost units; SMMSF achieves 4100 ms makespan, 210 kWh, 120 cost units, and 94 percent utilization. The three models incrementally address independent QoS scheduling, dependency-constrained workflows, and large-scale multi-objective scheduling.

Index Terms—Cloud computing, task scheduling, QoS, workflow scheduling, particle swarm optimization, clustering, multi-objective optimization.

I INTRODUCTION

Cloud computing provides shared processing, storage, memory, and network resources as services on demand, using an elastic virtualized infrastructure. The value of cloud computing is determined not only by the availability of resources but also by the efficiency with which submitted tasks are mapped to heterogeneous virtual machines. Therefore, task scheduling is a key cloud management problem, as allocation decisions affect response time, makespan, utilization, energy consumption, monetary cost, and service-level compliance [1, 2].

Cloud computing is a paradigm shift in computing that allows ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services). Resources are provisioned and released quickly with minimal management effort or service provider interaction. The National Institute of Standards and Technology (NIST) identifies five characteristics of cloud computing: on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service. Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) are service models that offer varying levels of abstraction and control to consumers, while deployment models include public, private, community, and hybrid clouds. In today's cloud data centers, virtualization technology is dominant to allow multiple virtual machines (VMs) to be executed on the same physical host. Virtualization allows resource isolation, dynamic resource allocation, and good resource utilization of the physical machine. However, the massive growth of cloud services, variety of workload and the size of data center pose great challenges to efficiently manage and schedule computational tasks [1].

“Cloud computing systems manage massive user requests, and task scheduling becomes a non-trivial problem.”

The successful running of user's tasks is also heavily dependent on the scheduling policy used by cloud provider. The task scheduling design includes the decisions for which tasks are assigned to which accessible VMs, resources to the assigned task for executing, execution order and computing, memory, bandwidth and storage resource for a certain task. Bad scheduling choices can increase execution time, and waste resources, energy, SLA violation and the higher cost of operation [2].

Scheduling of task in cloud computing is an activity of assigning the users' tasks to a particular set of VMs available in the cloud that optimizes one or more performance criteria. The scheduling problem can be described as follows: given a set of tasks $T = \{t_1, t_2, \dots, t_n\}$ and a set of virtual machines $VM = \{vm_1, vm_2, \dots, vm_m\}$, determine an allocation function $S: T \rightarrow VM$ to maximize a certain objective function with the consideration of constraints such as deadlines, priorities, dependencies, resource capacities and QoS requirements [3].

The complexity of cloud task scheduling arises from several factors:

Resource heterogeneity: Processing capability (MIPS), memory, bandwidth, and power consumption vary across virtual machines.

Tasks are not static: their arrival rate, size, and characteristics change over time.

Varied QoS demands: QoS issues may include constraints on execution time, cost, reliability and availability specified by users.

Workflow applications are made up of tasks with precedence constraints that need to be obeyed.

Multi-objective Property: Scheduling has to consider the trade-off among competing objectives such as makespan, energy, cost, and resource utilization.

"Task scheduling is an NP-hard problem where the computational complexity of mapping n tasks to m virtual machines is very high."

Because exact optimization methods become intractable for large-scale instances, researchers have turned to meta-heuristic optimization techniques, including Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Fruit Fly Optimization, and hybrid approaches, to find near-optimal solutions within reasonable computational time[4].

This research is motivated by the shortcomings of current scheduling techniques in meeting the complex requirements of modern cloud environments. First-Come-First-Served (FCFS), Round Robin, Min-Min, Max-Min and Suffrage, these traditional scheduling algorithms were mostly developed for homogeneous systems and single objective optimization. They frequently perform poorly in a heterogeneous, multi-objective, and/or dynamic cloud environment. In addition, many current methods handle all tasks in the same way, without taking into account workload attributes such as CPU requirements, memory needs, priorities, deadlines, and dependencies. This results in misallocation, load imbalance, and underutilization of resources. Energy consumption has also emerged as a critical factor, with large data centers being huge consumers of electricity and significant contributors to carbon emissions[5].

Hence, an adaptive, scalable and multi-objective scheduling scheme to: is in demand.

Dynamically prioritize tasks based on QoS parameters.

Group similar tasks and resources through intelligent clustering.

Respect task dependencies in workflow applications.

Simultaneously optimize multiple conflicting objectives.

Adapt to changing workload and resource conditions.

The importance of this study is that it may enhance efficient, scalable and sustainable cloud computing systems for the advantages of cloud providers (by decreasing operational expenditures and energy usage) and the users (better quality of service (QoS), diminished execution time, and trustworthy service supply).

The following challenges are central to this research:

Cloud workloads are dynamic and unpredictable by nature. The number, size and arrival of tasks are changing all the time. Likewise, virtual machine availability and performance may vary with virtualization overhead, network congestion, and co-resident load. Static parameter-based scheduling schemes fail to adjust to these dynamics and thus result in performance degradation [6].

The QoS needs for different jobs are also different. Some jobs are time-sensitive and have tight deadlines, but others are delay-tolerant. Tasks may also vary in priority according to user service level agreements (SLA). Great scheduling which meets and oversupplies the needs of critical jobs and deadlines and SLAs is an effective scheduling.

Several real-world cloud applications, including scientific workflows, big data analytics, and machine learning pipelines, are made up of inter-related jobs. No work is allowed to move ahead until all its predecessor tasks are finished. Not considering dependencies could cause wrong order execution, increased waiting time and workflow delays [7].

The cloud schedule algorithm should satisfy the multi-objective of balancing the cloud. For instance, in order to reduce the makespan, it is required to utilize more powerful and costly VMs, which elevates the cost and energy consumption. There could be a conflict between maximizing resource utilization and load balancing. Hence, to make framework more effective, multi-objective optimization techniques should be applied to obtain Pareto optimal trade-off.

Since cloud data center sizes can grow to comprise thousands or millions of VMs and tasks, algorithms need to be scalable. For large-scale systems, algorithms with high computational complexity become infeasible. Scalable frameworks that can manage a large number of tasks and resources with near-optimal performance are required[8].

Meta-heuristics are high-level problem-independent algorithmic frameworks that find sufficiently good solutions to complex optimization problems in a reasonable time. They are especially suitable for NP-hard problems for which exact methods are infeasible.

Particle Swarm Optimization (PSO) is a popular meta-heuristic for scheduling problems. PSO works with a population of particles, where each particle represents a potential solution, and is inspired by the social behavior of bird flocking and fish schooling. Particles move in the search space by updating their velocities according to their own best historical position and the global best position of the swarm.

Ant Colony Optimization (ACO): Inspired by the foraging behavior of ants, which use pheromone trails to guide their search.

Genetic Algorithm (GA): Inspired by natural selection, using crossover, mutation, and selection operators.

Fruit Fly Optimization Algorithm (FOA): Inspired by the food-searching behavior of fruit flies.

Crow Search Algorithm (CSA): Inspired by the intelligent behavior of crows in storing and retrieving food.

This research uses PSO and hybrid meta-heuristic strategies in an integrated scheduling framework.

This paper introduces a progressive scheduling framework composed of three interacting modules:

Quality of Service-Aware Task Scheduling Framework (QTSF): This module performs independent task scheduling for QoS-based clustering and priority-aware VM allocation.

Dependency-Aware Workflow Scheduling Model (DAWSM / DBTC): QTSF is extended to include workflow tasks that involve precedence constraints, using dependency-based clustering and PSO-based optimization.

Scalable Multi-Objective Meta-Heuristic Scheduling Framework (SMMSF): This module combines VM clustering, online resource monitoring, and multi-objective meta-heuristic optimization to handle large-scale cloud systems.

This paper makes four contributions. First, it integrates three complementary scheduling models into a unified architecture for observation, clustering, optimization, repair, and dispatch. Second, it defines seven algorithms in standard procedural form. Third, it proves finite termination and feasibility properties under explicit assumptions. Fourth, it reports the experimental evidence while distinguishing direct observations, derived comparisons, and limitations.



Fig. 1. Integrated research architecture.

II RELATED WORK

Relevant literature includes adaptive and sustainable multi-objective PSO, deadline-constrained scheduling, priority approaches, clustering, workflow optimization, reinforcement learning, prediction, and energy-aware scheduling [1]–[20]. Additionally, fruit-fly optimization, parallel PSO, trust-aware scheduling, QoS search, hybrid genetic scheduling, cloud-edge learning, resource-task clustering, nested PSO, reviews, and multi-objective metaheuristics extend the comparison [21]–[38]. This sequence sets the reference order used below.

Cloud task scheduling has attracted matured research during the last ten years with researchers advocating various kinds of optimizations including classical heuristics, clustering methods, etc. This survey paper critically reviews the most relevant modern approaches, it outlines their methodological underpinnings, their main achieved results and their limitations, which motivates the introduction of the adaptive meta-heuristic scheduling framework[11].

Workflow Scheduling Using Swarm and Evolutionary Optimization

The cloud workflow applications scheduling has been studied many times by nature inspired optimization algorithms. A Hybrid Collaborative Multi-Objective Fruit Fly Optimization Algorithm for Scheduling Workflow in Cloud Environment combines a cluster strategy with multi-objective fruit fly optimization to tackle two rivaling objectives of execution time and cost in workflow scheduling[12]. The obtained results showed better execution time and cost optimization than existing methods. Nevertheless, there is an important drawback found in this study which is the poor evaluation of scalability to large dynamic cloud environment that inhibits the applicability of the algorithm to real world cloud infrastructure where there exist high load of tasks and heterogeneous resource pools. This gap motivates the need for scalable scheduling algorithms that can maintain good performance when the size of tasks and resources grows [13].

Reinforcement Learning for Bi-Objective Scheduling

Machine learning, in particular reinforcement learning, application to cloud scheduling is becoming more popular. DDQN-TS: A New Bi-Objective Intelligent Scheduling Algorithm in Cloud Environment based on Double Deep Q-Network (DDQN)[14] framework to learn task response time and task completion rate at the same time. The reported results showed improvements in task response time and task completion rate[15]. However, the DDQN-TS algorithm has large training data complexity and computational complexity, which are the common drawbacks of deep reinforcement learning schemes[16]. The reliance on large-scale historical traces for training as well as the high cost of model convergence make such approaches less applicable to dynamic cloud environments, as they do not accommodate the determination of quick schedules[17].

Clustering-Based Scheduling for Workflow Applications

Clustering methods are now commonly used on tasks or resources to pre-schedule similar tasks with similar resources and reduce the complexity of the mapping problem[18]. The Resource and Task Clustering Based Scheduling Algorithm for Workflow Applications in Cloud uses K-means++ clustering and list scheduling techniques to achieve better makespan and clustering efficiency than the HySARC algorithm[19]. While the combination of clustering and list scheduling improved performance, the method does not adequately handle dynamic workload variations and energy consumption, which restricts its use in energy-efficient and dynamically changing cloud data centers. This directly highlights the need for clustering techniques that can simultaneously consider QoS parameters, resource heterogeneity, and energy efficiency[20].

Dependency-Aware Workflow Clustering and Balanced Scheduling

Another important aspect to consider when scheduling workflow tasks is the structural dependencies between them, as these must be preserved throughout the scheduling process. The paper, Balanced Scheduling of Distributed Workflow Tasks Based on Clustering, proposed the Runtime Balance Clustering Algorithm (RBCA) and the Dependency Balance Clustering Algorithm (DBCA) which aim to reduce the overhead of scheduling and the execution time of workflows[21]. The dependency-aware clustering approach clustered tasks based on their execution dependencies, thus maintaining workflow semantics. However, the approach is mainly focused on dependency and runtime balancing, and does not take into account cloud energy efficiency, which is an important emerging objective in modern sustainable cloud computing[22]. This limitation underscores the importance of scheduling frameworks that simultaneously optimize dependency preservation, runtime performance, and energy consumption [23].

Credit-Based Scheduling with K-Means Clustering

Clustering has been combined with economic or priority-based mechanisms as a potential strategy to enhance QoS and resource utilization [24]. A Credits Based Scheduling Algorithm with K-Means Clustering uses credit-based scheduling and K-means clustering to group tasks and virtual machines, and is claimed to enhance QoS and resource utilization in cloud scheduling. However, the method is not effective at handling heterogeneous workloads, as uniform handling of tasks with

varying computational needs can result in clusters that are not optimally balanced and a less than ideal VM allocation. Then, An Improved Task Allocation Strategy in Cloud Using Modified K-Means Clustering Technique extended this work by adding credit-based task prioritization and showed gains in makespan and computational cost in CloudSim simulations. However, these techniques are only evaluated on small-scale distributed cloud systems, which limits empirical validation of the approach's scalability [25].

Energy efficiency has become one of the key issues of cloud resource management, and this has led to research into energy-aware VM placement strategies. Energy Aware Cluster Based Optimal Virtual Machine Placement in Cloud Environment adopted hierarchical clustering based VM placement technique with Crow Search Algorithm, minimizing energy consumption, SLA violation, and maximizing resource utilization. Nevertheless, the VM placement solution does not consider a dynamic task to the meta-heuristic crow search, which is concentrated on static placement decision. This is an important differentiation as VM placement and task scheduling are, in fact, the same type of problem at different temporal scales of resource management in cloud systems [26].

The optimization of large-scale service processes under QoS constraints has been addressed through parallel skyline computation and meta-heuristic techniques. Parallel Optimization of QoS-Aware Big Service Processes with Discovery of Skyline Services integrated parallel skyline service discovery with meta-heuristic optimization using Apache Spark, demonstrating improved execution plan selection under QoS constraints[27]. However, the approach exhibits high computational complexity for extremely large service sets, which limits its practicality for real-time scheduling in massively scaled cloud environments. This limitation reinforces the necessity for scalable approximation algorithms and heuristics that can deliver near-optimal solutions within acceptable computational budgets.

Synthesis and Identification of Opportunities

The surveyed literature shows that no single existing method is suitable for all four aspects of the challenge: dynamic resource awareness, efficient task grouping according to workload characteristics, simultaneous multi-objective optimization, and scalability in large-scale heterogeneous systems[28]. Fruit fly optimization and reinforcement learning-based workflow scheduling algorithms enhance certain metrics but are not scalable and have high computational overhead. Cluster-based approaches enhance grouping efficiency but often fail to account for dynamic workload variations, energy consumption, or heterogeneous workload processing. Dependency-respecting clustering techniques maintain the workflow structure but disregard the energy consumption. Credit-based and energy-efficient VM allocation methods cope with profit maximization and energy minimization but they are not capable of dealing with real-time task scheduling and/or heterogeneous workload. QoS-aware service optimization approaches encounter computational hardness barriers at scale[29].

Hence, a very promising research direction is to develop an integrated adaptive meta-heuristic scheme, which integrates QoS-aware task clustering, dependency-aware workflow scheduling and scalable multi-objective optimization in a unified cloud scheduling framework. We aim to fill this gap in the literature by proposing the Quality of Service-Aware Task Scheduling Framework (QTSF), the Dependency-Aware Workflow Scheduling Model (DAWSM), and the Scalable Multi-Objective Meta-Heuristic Scheduling Framework (SMMSF), which are designed to overcome specific limitations of the current literature and provide a more integrated scheduling solution for large-scale cloud computing environments[30].

$$ETC_{ij} = L_i / MIPS_j \quad (2.1)$$

$$M = \max_j (\sum_i x_{ij} ETC_{ij}) \quad (2.2)$$

$$v_i(t+1) = w v_i(t) + c1 r1[pbest_i - x_i(t)] + c2 r2[gbest_x_i(t)] \quad (2.3)$$

$$\mu_{ik} = 1 / \sum_j (|x_{i-c_k}| / |x_{i-c_j}|)^{2/(m-1)} \quad (2.4)$$

$$P_j = P_{idle,j} + (P_{max,j} - P_{idle,j}) U_j \quad (2.5)$$

Scalability implication: Fcfs and round robin heuristics provide low decision overhead, but do not consider heterogeneous execution costs. In the context of the research being proposed, their simplicity becomes a limitation when QoS attributes are different. Clustering and staged evaluation reduce the number of pairwise task-to-resource comparisons prior to meta-heuristic refinement. Relevant state variables are measured before allocation, converted into normalized indicators, and input into the decision function. This process of measurement and optimization are kept separate, and the basis of an assignment is transparent. It also allows for controlled ablation: the part can be removed, its metric change noted, and the benefit of the part compared to the rest. This understanding also prevents a nominally fast VM from being chosen for heterogeneous cloud resources if queue length, memory pressure, dependency delay, or energy cost causes another VM to be more optimal. The Weighing problem thus can be interpreted as a disciplined form relating the research gap, the algorithmic action, and the observed performance consequence [9, 12].

Scalability Implication: Min-min and max-min scheduling rely on the estimated completion time as a selection criterion. In the context of the proposed research, they are possibly creating resource starvation and do not explicitly maintain workflow dependencies. Clustering and staged evaluation discard a portion of pairwise task-to-resource comparisons prior to meta-heuristic refinement. The relevant state variables are observed prior to allocation, they are converted into normalized indicators and passed to the decision function. This process distills separation of measurement from optimization, and makes the "why" of an assignment traceable. It also allows for controlled ablation: the part can be taken out, the metric change can be recorded, and its contribution can be compared with that of the entire configuration. For heterogeneous cloud resources, this

interpretation eliminates the selection of a nominally fast VM when another VM is preferred by this criterion due to long queue, excessive memory pressure, high dependency delay, or large energy cost. The obtained results advocate the disciplined link between the problem statement, the algorithmic scheme, and the performance outcome witnessed [10, 13].

Scalability implication: k-means task grouping leads to less individual scheduling tasks. In the context of the research being proposed, the feature scaling, centroid initialization, and cluster count influence the result. Clustering and staged evaluation reduce the number of pairwise task-to-resource comparisons prior to meta-heuristic refinement. Relevant state variables are observed before allocation, converted to normalized indicators, and input to the decision function. This workflow splits measurement from optimization, and allows the assignment to be explained. It also enables controlled ablation: the component can be removed, its metric change recorded, and its contribution compared to the complete configuration. This approach to heterogeneous cloud resources avoids selecting a nominally fast VM when queue length, memory pressure, dependency delay, or energy cost cause a different VM to be more optimal. The evidence thus obtained provides a disciplined link between the research gap, the response algorithm and the performance reported [11, 14].

Scalability implication: Particle swarm optimization explores a large mapping space using personal and collective experience. In the context of the research being proposed, constraint repair is needed when particle movements violate precedence or capacity. Clustering and staged evaluation reduce the number of pairwise task-to-resource comparisons prior to meta-heuristic refinement. Relevant state variables are observed before allocation, converted to normalized indicators, and input to the decision function. This sequence decouples measurement from optimization and makes the motivation for an assignment traceable. Among other things, this also makes a controlled ablation possible: the module can be ablated, the difference in metric recorded, and the contribution compared to the full system. For the case of heterogeneous cloud resources, this interpretation avoids the selection of a nominally fast VM when such VM is de facto slow due to a long queue on the host, high memory pressure, inter-VM dependency delay, or high energy cost. The upshot is a more evidence-based disciplining of the research gap, to the algorithmic response, to the reported performance outcome [12, 15].

TABLE I. Comparative Taxonomy

Family	Benefit	Limitation
Classical heuristics	Low overhead	Static narrow objective
Clustering	Reduced search	Feature sensitivity
PSO ACO GA	Broad search	Convergence cost
Workflow	Precedence	Communication cost
Learning	Adaptation	Training and shift
Multi-objective	Trade-offs	Preference selection

III SYSTEM MODEL AND INTEGRATED FRAMEWORK

Cloud task scheduling is considered as a constrained mapping problem to assign tasks to heterogeneous VMs. The decision is difficult due to dynamic arrivals, heterogeneous resource capacities, user-specified service requirements, workflow dependencies, and multiple provider objectives. Prior schedulers are often designed to optimize some limited objective in a resource static view, or they handle independent tasks and dependent tasks using the same decision rules, or they suffer severe searching overhead when workloads grow. The research challenge is to develop an adaptive scheduling framework that maintains QoS differentiation, respects workflow dependencies, trades off time, energy, cost, and utilization, and scales computationally with increasing task and VM populations; none of the prior work adequately address all of these aspects [3, 7, 32, 33].

$$S^*: T \rightarrow VM = \arg \min_S J(S) \quad (1.1)$$

$$J(S) = a M(S) + b E(S) + c C(S) + d[1-U(S)] \quad (1.2)$$

Several scheduling algorithms have ranked tasks according to a single dominant attribute, or considered resource information available only at the moment of task submission. These decisions do not fully reflect the combined effects of task length, priority, deadline, CPU demand, memory demand, current VM load, as well as residual capability. This raises the challenge of a QoS aware method, which takes normalized task properties, current resource capacity, and load-repair decisions [1, 4, 6] into consideration.

The vast majority of the scheduling literature is under the assumption of independent tasks. Such operations on workflows can violate precedence, postpone child jobs, and raise communications overhead of inter-dependent jobs. There is a research void for a clustering & meta-heuristic search method that considers the directed-acyclic-graph structure, generates a legal topological order, and repairs infeasible candidate solutions in the process of search [7, 8, 33].

Time-oriented schedulers may reduce makespan while increasing energy use, monetary cost, or resource imbalance. Multi-objective methods address this conflict, yet fixed weights, large task-to-VM search spaces, and weak runtime adaptation can limit their scalability. A gap exists for a dynamic framework that clusters resources, normalizes conflicting metrics, adjusts exploration pressure, and demonstrates stable behavior over increasing workload sizes [2, 5, 32].

To discover the QoS aware Task Scheduling Framework (QTSF) that groups the tasks based on the normalized service and resource parameters, the tasks are prioritized based on the service requirement, then tasks are mapped to the suitable VMs and finally performs load imbalance.

To propose a Dependency-Aware Workflow Scheduling Model (DAWSM) that performs execution of workflow graphs, constructs dependency-aware task clusters, maintains precedence by means of topological sorting and repairing, and optimizes workflow completion time by exploiting dependency-based PSO.

To propose a Scalable Multi-Objective Meta-Heuristic Scheduling Framework (SMMSF) which adopts dynamic VM monitoring and clustering to maximize the makespan, energy consumption, monetary cost, and resource utilization for large workloads.

The integrated framework combines three complementary scheduling models. QTSF applies normalized QoS clustering, capability-based VM mapping, and load-balance repair. DAWSM combines dependency-based clustering, precedence-feasible particle decoding, and constraint repair. SMMSF adds dynamic VM clustering and jointly minimizes makespan, energy consumption, monetary cost, and utilization loss under large workloads.

Let x_{ij} be one, if task i is assigned to VM j . The assignment equality $\sum_j x_{ij} = 1$ for each task i together with the CPU, memory, bandwidth, availability, and workflow constraints define the feasibility. $ETC = \text{task length} / \text{VM MIPS}$. Completion Time adds machine ready time, makespan is maximum finish time, energy is a function of power that depends on utilization, and cost accumulates execution cost. The metrics are normalized prior to weighted aggregation.

The combined framework decouples the measurement from the optimization. Task descriptors and current VM state are observed first. QTSF applies service and demand properties to independent jobs. DAWSM adds graph topology and communication. SMMSF adds monitored resource clusters and four-objective fitness. All models maintain hard capacity constraints and a repair step, so a squared-up-but-infeasible candidate is not dispatched.

IV QOS AWARE TASK SCHEDULING FRAMEWORK

A Proposed Model

Quality of Service has become an important factor in cloud schedule as it defines the service assurance desired by users. QoS parameters are execution time, response time, throughput, deadline fulfillment, resource utilization, reliability and load balancing. The proposed Quality of Service-aware Task Scheduling Framework (QTSF) overcomes these limitations through some novel characteristic, such as QoS evaluation, task clustering and priority-based virtual machine allocation in a single task scheduling framework. Instead of scheduling each task as a single unit, by considering task features, the framework performs feature analysis, calculates QoS scores based on weighted parameters and groups similar tasks into clusters. Then, scheduling decisions are performed at the cluster level to reduce the complexity of scheduling and improve the scalability. Priority task groups are assigned to appropriate virtual machines VM on the basis of processing capacity and resource availability.

The designed framework is developed and tested with CloudSim simulation toolkit. Electrical performance evaluation of the proposed technique is presented against K-Means integrated with Ant Colony Optimization

(ACO), Particle Swarm Optimization (PSO) and traditional scheduling methods. In terms of makespan, balance load and resource utilizations, we evaluated the performance. The results obtained reveal that the QoS-based clustering approach can significantly improve the efficiency of scheduling with minimum execution time and maximum overall cloud performance.

The traditional scheduling techniques, First they are rely on static parameters too much. It is believed that Furthest-First-Task-Ordering (FFTO), Highest-Task-First (HTF), and Round-Robin Based Incomplete Chaining (RRBIC) c consider only one component of a task arrival sequence or length in task selection, for example, FCFS, Min-Min, Max-Min, and round-robin. Second, most existing approaches regard tasks as atomic entities and do not consider the multi-dimensional aspect of QoS. In practice, users run tasks having different priorities, deadlines, and resource demands. A scheduler that treats all these dimension equally cannot offer differentiated service. Third, many of the previously cluster-based schedulers only cluster tasks and do not fine-tune clusters according to actual resources demands. This can form clusters that are statistically consistent in the QoS feature space but are computationally infeasible given the underlying VM configurations, resulting in either resource over-provisioning or resource contention.

Secondly, task-to-VM assignment is often independent of load-balancing feedback. Even if a good initial mapping is found, over time some VMs may become heavily loaded while others remain idle, resulting in increased makespan and reduced performance as perceived by the user.

QTSF addresses the problem defined as follows. We are given a set of independent tasks , where each task is defined by its length, priority, deadline, CPU requirements, and memory requirements, and a set of heterogeneous VMs, each with CPU, RAM, and bandwidth capacities, and we want to find a mapping that (i) minimizes makespan, (ii) maximizes resource utilization, and (iii) keeps VM loads balanced, subject to the QoS priority ordering defined by task length, priority, and deadline.

QoS-aware scheduling studies commonly employ priority rules, credit functions, clustering, or population-based optimization. Priority rules are cheap, but a priority quality can never represent the interaction between deadline, task size, and resource requirements. K-means schedulers decrease the overhead of assignment by bundling similar tasks, but some implementations cluster only task length and do not verify that the clusters formed correspond to the actual VM capabilities. ACO and PSO schedulers consider a wider search space for assignment, but their convergence cost is exponentially increasing with the number of task-to-VM pairs. QTSF lies in the space of these families: multidimensional clustering initially reduces the search space, priority scoring ensures service differentiation, and capability matching and load repair regulate resource disparity.

The QTSF framework is formulated as a two-stage scheduling pipeline: (1) QoS-aware task clustering, and (2) QoS-based task scheduling and VM assignment. This division is deliberate. Clustering simplifies the complexity of search space by clustering similar tasks before the allocation phase, while scheduling phase assigns a task cluster (tasks in each cluster have the equal demands) to the VM whose capability profile is the nearest to its resource demand. The two stages are connected through a common notion of QoS priority and an explicit load-balancing correction loop.

Initially, the clustering stage is to calculate a scalar QoS for each task. The score is defined as a linear weighted combination of three attributes of the task:

where are non-negative coefficients representing the relative significance of the task length, the priority, and the deadline. This formulation is simple, yet flexible: depending on the values of the weights, the framework can be configured to focus on short urgent tasks, long high-priority tasks, or tasks that are critical to the deadline as dictated by the service-level agreement (SLA). Once the QoS scores are obtained, for each user, the tasks are grouped with a Euclidean-like squared-difference similarity measure:

The clustering objective is to minimize the within-cluster QoS dissimilarity:

This is conceptually the same as K-means clustering on a one-dimensional QoS feature space. The use of squared distance is natural in leastsquares optimization and permits easy iterative refinement.

QTSF also adds a resource aware refinement step to stop purely QoS driven clusters from becoming isolated from the underlying resource infrastructure. The resource requirement of each task is computed as:

and clusters are refined further by minimizing the square difference in resource demands:

This refinement at the second level is a key architectural strength. This means tasks that are bunched together are not only near each other in their user centric QoS parameters but are also close in their resource usage, which leads to a more coherent framework when VM assignment happens later.

Phase 2: QoS-Based Task Scheduling and VM Allocation

The clusterized task set and the VM set are received by the scheduling stage. First, it calculates a priority score for each task:

where the estimated execution time on a reference VM is:

Tasks are then arranged in descending order of QoS score, so that high priority, deadline sensitive or long running tasks are handled first in allocation. This ordering is significant, as greedy schedulers that prioritize high-priority tasks often obtain superior VM slots before lower-priority options are exhausted.

Each VM is characterized by a composite capability score:

and each task is assigned to the VM that has the minimum absolute difference between capability and requirement:

This best-fit strategy seeks to prevent both over-provisioning (assigning a small task to a powerful VM) and under-provisioning (assigning a large task to a weak VM). The framework evaluates load balance after initial assignment using the metric:

where denotes the utilization of VM . The goal is to minimize . If the current mapping is considered imbalanced, tasks are moved from overloaded VMs to underloaded ones. This iterative refinement continues until all tasks are scheduled.

QTSF is a hybrid methodology that leverages cluster analysis (dimensionality reduction and workload grouping), greedy priority scheduling (computational efficiency), and load-balancing feedback (dynamic correction). Algorithm

1: QoS-Aware Task Clustering

QTSF's first algorithmic component is an explicit formalization of the clustering phase. The document describes the inputs, outputs, and procedural steps as follows:

The QTSF framework is depicted in Fig. 2. First, the CloudSim parameters specifying the datacenter, hosts, virtual machines, cloudlets, broker policy, processor capacity, memory, bandwidth, and simulation clock are initialized. This phase creates a controlled and reproducible resource environment before any scheduling decision is made. The CloudSim entities are then instantiated through the broker layer. The broker collects task definitions, monitors VM status, invokes clustering and ranking, allocates tasks, and records the scheduling outcomes.

The task-profile input at the right of the framework provides the broker with task length, priority, deadline, and cost information. Task length is the required amount of computation, priority is the relative service importance, deadline is the urgency, and cost is the economic impact of using a specific VM. These values are normalized before being combined as their original units and ranges are different. The normalized task length, priority, and deadline urgency are the QoS score in Equation (3. 1) while estimate execution time and resource requirement are then involved to obstruct a high priority task being assigned to an incapable machine.

Within the datacenter-broker boundary, a task set $T_1, \dots, T_n$ is submitted to the QoS-based clustering method. Comparable QoS tasks are grouped in the same operational cluster, and resource requirement refinement also distinguish tasks that are ostensibly alike in service priority, but are very different in CPU or memory requirements. The generated clusters decrease the number of disjointed task to VM comparisons, and then the scheduler can apply different treatments on urgent, calculation consuming and regular sets of tasks. Hence, the framework transforms a bland cloudlet queue into a pack of scheduling units each carrying definite service semantics.

The VM scheduling module employs priorities to decide which ready tasks are to be considered first for allocation, but the actual assignment is limited by the VM resource availability, residual resource, estimated execution time, etc., and the usability and current usage of the employed VMs as well. For each potential pair, the broker assesses the execution time by Equation (3.5), determines the capability dissimilarity, and identifies a suitable VM by the integrated dissimilarity and completion metric in Algorithm 3.2. This explains why the label VMs scheduling with priorities in the framework does not mean priority-based only allocation, priorities govern the order in which tasks are dispatched and availability and capability dictate when and where they are placed.

The final stage schedules tasks with currently highest cluster scores. After each VM placement, the VM utilization is recalculated and the load spreading in Eq. (3.6) is calculated. If the load-imbalance criterion is violated, the repair scheme transfers a candidate task from the over loaded VM to a less loaded under loaded VM as long as this transfer will not introduce a larger completion time cost. The result of Figure 3.1 is therefore not a mere priority queue, but rather a QoS-differentiated, capability-aware, and load-repaired task schedule.

$$Q_i = w_1 TL_i + w_2 TP_i + w_3 TD_i \quad (3.1)$$

$$\sum_{k=1}^3 w_k = 1$$

$$d(i, j) = (Q_i - Q_j)^2 \quad (3.2)$$

$$RR_i = CPU_i + MEM_i \quad (3.3)$$

$$P_i = \frac{Q_i}{ET_i} \quad (3.4)$$

$$ET_{ij} = \frac{L_i}{MIPS_j} \quad (3.5)$$

$$LB = \max_j U_j - \min_j U_j \quad (3.6)$$

Equation (3.1) aggregates\ t three distinct service characteristics into a single QoS measure. Length, priority, and deadline of the task are normalized to the same scale before being aggregated. The coefficients w_1 , w_2 and w_3 are assumed to be non-negative and their summation is unity, thus every coefficient corresponds to a relative scheduling preference over its associated attribute. A greater deadline-related weight makes the scheduler more urgency-aware; a greater length-related weight induces more differentiated treatment for computationally intensive jobs. Weights can be determined based on SLA policy or by using analytic hierarchy method, or based on validation experiments.

Equation (3.2) provides the intra-cluster dissimilarity defined in the QoS grouping access. Dis-E contrasts with location-scale models of the distance-based model, and accounts for the more severe penalty for distant point pairs, as manifest in nearest-neighbor models. Equation (3.3) shows the concise resource-demand index. When implementing, CPU and memory terms should be normalized or weighted before addition since they have different dimension. This second-level distance also allows to separate tasks that are equally urgent but rival in resource demand.

Service importance is related to machine speed by (3.4) and (3.5). The priority-to-time ratio for an important item that can be finished in little time increases, so that the scheduler can consider both urgency and feasible due date. The ETC value of (3.5) is calculated for all task-VM pairs, which gives rise to the execution-time matrix used in capability matching. Equation (3.6) is the measure of utilization concentration. A number close to zero indicates similar loading, while a large value indicates the operator need to be repaired. This measure of dispersion is cheap and can be calculated after each assignment.

For QTSF, $K=4$ QoS task clusters are used. Min-max normalized features receive weights $w_1=0.40$ for task length, $w_2=0.30$ for priority, and $w_3=0.30$ for deadline urgency. K-means terminates after 100 iterations or when centroid displacement falls below 10^{-4} . VM mapping uses mismatch weights $\eta_1=0.60$ and $\eta_2=0.40$, with a load-imbalance repair threshold $\theta_{LB}=0.20$. Each configuration is evaluated over 30 simulation runs using random seed 50 for repeatability.

priority alone is not determining VM allocation. The mapping decision is considered based on the QoS score, Length of task, deadline urgency, CPU, memory, Bandwidth demand, Estimated execution time, Residual VM capacity, Current utilization, Availability, Completion time, Load-imbalanced.

priority determines the order in which ready tasks are served; and feasibility and capability mismatch determine the VM.

Task priority density is $P_i=Q_i/\min_j(ET_{ij})$. Q_i is a weighted combination of normalized task length, service priority, and deadline urgency. contains information from the particular machine-dependent expected execution time ET_{ij} . A high P_i corresponds to a task with a strong QoS requirement and a short expected completion time; however, it must still respect the capacity and availability constraints.

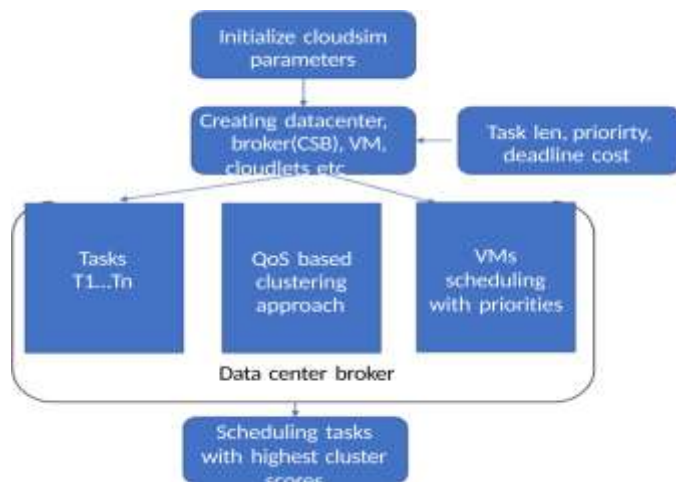


Fig. 2. QTSF scheduling framework.

Algorithm 1. QoS Aware Task Clustering

Input: Task set T with length, priority, deadline, CPU, and memory requirements; cluster count K ; weights w_1 , w_2 , and w_3 ; tolerance epsilon; maximum iterations R .

Output: QoS task clusters $C_1, C_2, \dots, C_K$.

1. Normalize task length, priority, and deadline urgency to a common interval.
2. For each task t_i , calculate its QoS score Q_i using (3.1) and its resource requirement RR_i using (3.3).
3. Initialize K centroids from distinct tasks.
4. Assign every task to the nearest centroid using the QoS dissimilarity in (3.2), while retaining resource-demand compatibility.
5. Recompute every centroid from the tasks assigned to its cluster.
6. Repeat Steps 4 and 5 until centroid movement is below epsilon or R iterations are completed.
7. Sort the resulting clusters and their tasks by priority density P_i in (3.4), then return the clusters.

Algorithm 2. QoS Based VM Allocation and Load Repair

Input: Ordered task clusters C ; virtual machines VM ; residual capacities; imbalance threshold θ_{LB} .

Output: A feasible task-to-VM mapping S .

1. Measure each VM capability, availability, queue state, and current utilization.
2. Select the highest-ranked unscheduled task from the ordered clusters.
3. For every feasible VM, compute execution time ET_{ij} using (3.5) and evaluate capability mismatch and predicted completion time.
4. Assign the task to the feasible VM with the smallest combined mismatch and completion criterion.
5. Update the selected VM residual capacity, ready time, and utilization.
6. Calculate load imbalance LB using (3.6).
7. If LB exceeds θ_{LB} , move an eligible task from the most-loaded VM to a feasible less-loaded VM when the move does not increase the completion penalty.
8. Repeat Steps 2-7 until every task is assigned; return S .

B Formal Analysis

Proposition 1. With cluster count and iteration bound being both finite, QoS clustering stops after a finite number of iterations.

Proof. Each iteration is a finite assignment and update of centroid. Stability or tolerance aborts the procedure at an early stage; otherwise, the finite counter hits its bound. $\square$

Proposition 2. If every task has a accessible VM and repair maintains unique assignment, QTSF schedules every task once.

Proof. Initial assignment exists. Repair takes out a task prior to adding it to one feasible alternative. Induction on the number of finite placements and repairs maintains the invariant. $\square$

For n task, m VMs, K clusters and r clustering iterations, clustering is $O(rnK)$, execution-time evaluation and mapping is $O(nm)$, and sorting is $O(n \log n)$. A bounded repair loop introduces a finite scan of tasks and resources. The entire execution-time matrix needs $O(nm)$ memory, streaming can alleviate the need for so much memory.

V DEPENDENCY AWARE WORKFLOW SCHEDULING MODEL

A Proposed Model

DAWSM extends QoS-aware scheduling to workflow applications by incorporating task dependencies directly into allocation decisions. It clusters related tasks through dependency analysis and applies metaheuristic optimization while preserving legal execution order and reducing workflow delay. Dependencies are therefore part of candidate generation, decoding, fitness evaluation, and repair rather than an after-the-fact validation step.

DAWSM framework consists of two algorithmic pillars: Dependency-Based Workflow Task Clustering (DBTC) and Dependency-Aware Scheduling with Particle Swarm Optimization (DAS-PSO). DBTC clusters tasks that have successor or predecessor relations so that closely related tasks are scheduled/placed together. DAS-PSO explores the space of viable task-to-VM mappings guided by a fitness function that considers the execution time and the dependency-induced delay, and a constraint handling procedure ensures that all the precedence relations are satisfied.

The rationale for dependency-aware scheduling is the fact that today's cloud applications are more and more workflow-oriented. Examples are genomic pipelines, weather simulations, image processing chains, and machine-learning training workflows.

processing repair step. This provides an impetus for a model that integrates/imbeds dependency structure within the inference and optimization steps at inception.

In particular, the issue solved by DAWSM can be formally defined as follows. Represent a workflow as a DAG, where T is the set of tasks and E is the set of directed dependency edges. Each task has a computational length and resource demand, and each edge imposes the constraint

For a heterogeneous set of VMs, the goal is to find a schedule that assigns every task to a VM and a start time such that all precedence constraints are met and a weighted sum of execution time and dependency delay is minimized. This problem is NP-hard in the general case, which is why meta-heuristic search is often used.

Workflow schedulers, as opposed to independent-task schedulers, must maintain precedence and data-transfer relations. List-based approaches order tasks according to upward or downward priority, clustering models group connected vertices, and PSO or ACO methods explore the mapping space. DBCA enhances dependency balancing, but dependency grouping is not sufficient to choose the optimal heterogeneous VM mapping. PSO can optimize execution time, but a particle representation may become infeasible if a child is moved before its parent. DAWSM integrates dependency-based clustering with a discrete PSO representation and a repair operator, which ensures structural feasibility is preserved throughout the optimization process, rather than only being verified once convergence is reached.

The DAWSM approach is built around a three-stage pipeline: dependency extraction, dependency-based workflow clustering, and PSO-driven schedule optimization with constraint enforcement. Each stage is based on the output of the previous one and processes the raw DAG to an optimized and executable task-to-VM mapping.

For every job within the workflow, the scheduler retrieves its dependence set:

This set represents the local neighborhood of each task in the DAG. The scheduler then calculates a dependency similarity for two tasks, t_i and t_j , based on the intersection between their successor sets:

The motivation is that tasks which have more common successors are more likely to have output data consumed by the same lower-level tasks. When these tasks are co-located or scheduled closely, these actions minimize inter-VM data transferring and enhance cache/data locality. The consolidated dependency score for a task is:

Stage 2: Dependency-Based Workflow Task Clustering (DBTC)

With the dependency scores, DBTC splits the workflow into clusters of dependent tasks. The clustering objective is:

promote clustering such that elements in the same cluster have stronger dependencies). By performing clustering, DAWSM also diminishes the effective scheduling problem size and guarantees that closely linked tasks are handled as single entities.

Stage 3: Dependency-Aware PSO Scheduling (DAS-PSO)

After the clustering procedure, DAS-PSO seeks the optimal task-to-VM placement. Each particle in the swarm represents a candidate schedule. Both total execution time and dependency delay are explicitly penalized by the fitness function:

The velocity and position updates follow the canonical PSO equations:

This component corrects or penalizes particles that breach precedence, thus guiding the swarm to feasible schedules. The algorithm runs until convergence, updating personal best () and global best () positions, and returns the final schedule.

In sum, DAWSM prescribes a coherent methodology for converting dependency structure into scheduling gain. The model also achieves reduction in communication overhead and risk privacy breach with a dependency aware heuristic fitness function by performing clustering on dependency overlap.

Fig. 3 presents the complete DAWSM processing sequence. Scheduling begins when a workflow request is accepted and its tasks are collected. Unlike independent-cloudlet scheduling, the input is a connected application in which a task may depend on the completion and data output of one or more parents. Each task record therefore contains computation length, resource demand, predecessor and successor identifiers, communication volume, and any applicable deadline or cost.

Dependency discovery parses the workflow specification for all parent-child relationships. The discovered relationships form the DAG $W = (V, E)$ defined in equation (4.1), where vertices represent tasks and directed edges represent the order of precedence as well as possible communication. Building the graph is crucial since a length-based scheduler might issue a child before the inputs it needs are ready. Topological analysis identifies an illegal cycle, constructs the legal execution skeleton and generates a ready-task set, which at that point contains only vertices whose predecessors are already finished.

DCTU further employs a dependency-based clustering phase that clusters tasks according to common predecessor and successor structure. The correlation measure of Equation (4.2) lifts two tasks which share many downstream vertices up, and (4.3) sums up the dependency strength of a seed candidate. Cluster-size and resource limits are imposed so that a dense region will not grow too large for effective optimization. By keeping strongly-related tasks in the same scheduling container, the model can reduce the inter-cluster communication, and the local precedence information of clusters can be kept during the searching process.

Meta-heuristic scheduling considers the dependency clusters, their legal internal orders, and the specifications of the heterogeneous VM types. DAWSM represents a candidate with VM assignments along with precedence-feasible priority keys. The particle-swarm update in Equation (4.5) searches over alternative mappings, and the fitness function in Equations (4.4) and (4.6) 四个方面都有良好表现：运行时间、依赖延迟、算量开销及负载不平衡。候选映射仅在恢复了依赖顺序之后才被解码，这阻止了优化器通过违法 workflow 顺序来提高数字适应度。

The VM scheduling is to allocate the task or cluster unit after scheduling to the available machine's units. The dependency-constraint check then applies to all edges (including those which may be delayed by executing the parent on one VM and the child on a different one). A violated edge is fixed by delaying the child or by changing the child's assignment to a different VM; a violated overloaded machine is fixed by its compatible alternative placement. The tasks are run only when these checks complete successfully. The performance-evaluation section logs dependency satisfaction, workflow execution time, clustering efficiency, and computational cost and, then the scheduling ends. Hence, any arrow in Figure 4.1 denotes a feasibility-preserving transformation, not a merely sequential refinement of code.

$$W = (V, E), \quad (u, v) \in E \Rightarrow FT_u \leq ST_v \quad (4.1)$$

$$cor(i, j) = \frac{|succ(i) \cap succ(j)|}{|succ(i) \cup succ(j)|} \quad (4.2)$$

$$DS_i = \sum_j cor(i, j) \quad (4.3)$$

$$F(S) = a ET(S) + b DD(S) + \lambda \Phi(S) \quad (4.4)$$

$$v_p(t+1) = wv_p(t) + c_1 r_1 [pbest_p - x_p(t)] + c_2 r_2 [gbest - x_p(t)] \quad (4.5)$$

$$F_2(S) = 0.40ET_n(S) + 0.30DD_n(S) + 0.20CC_n(S) + 0.10LI_n(S) \quad (4.6)$$

Equation (4.1) treats the workflow as a directed acyclic graph and specifies the basic precedence constraint. An edge (u, v) means that task u must be completed before task v can be started. In the presence of data transfer, the right-hand side is extended to the communication time of the VMs allocated to the two tasks. A topological order is unique only when the graph is acyclic, and this order represents the legal skeleton upon which the optimizer develops candidate schedules.

Equation (4.2) is a Jaccard-like similarity function over the successor sets. Two tasks sum high for the parallelism with many of the same downstream tasks. The union in the denominator makes the score normalised between zero and one and prevents bias towards tasks with many successors. Predecessor similarity and edge weights may be introduced analogously. Equation (4.3) summarizes pairwise relations into a dependency strength that is employed in DBTC cluster formation. A cluster-size and a resource-capacity constraints prevent an unbalanced clustering that matches the entire graph connectivity to become a single cluster.

Equation (4.4) computes the value of a schedule as a function of its execution time, dependency delay, and violation penalty. The penalty $\Phi(S) = 0$ for a schedule without violations, and increases with each precedence or capacity constraint violation. A large λ sufficiently discourages infeasible particles, but the repair phase retains its usefulness by bringing particles back to the legal search domain. Equation (4.5) depicts the continuous PSO update. DAWSM converts the resulting position to VM assignments and priority keys, and to maintain topological validity after every move.

DAWSM evaluates each candidate schedule using normalized execution time ET_n , dependency delay DD_n , computational cost CC_n , and load imbalance L_n , as defined in (4.6). ET_n represents workflow makespan, DD_n is the sum of precedence-induced waiting times, CC_n is the total VM execution charge, and L_n measures utilization dispersion. PSO uses a swarm size of 40, an inertia weight that decreases from 0.90 to 0.40, $c_1=c_2=2.0$, and 100 iterations. It terminates after 10 non-improving iterations or when the fitness change falls below 10^{-5} .

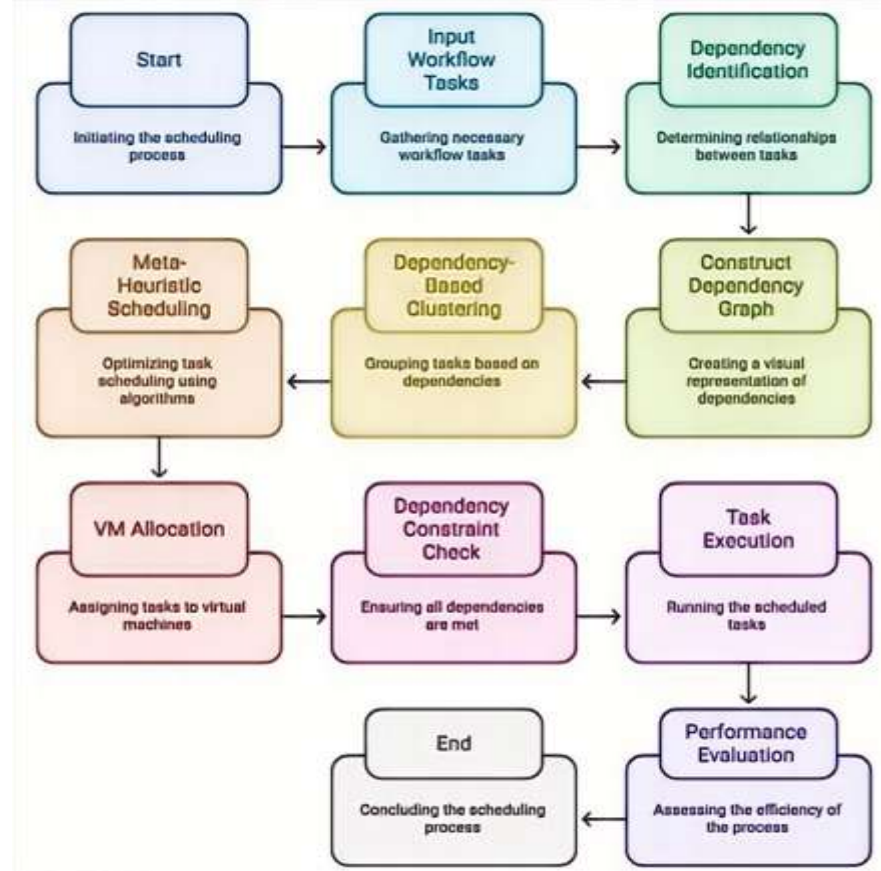


Fig. 3. DAWSM workflow scheduling framework.

Algorithm 3. Dependency Based Workflow Task Clustering

Input: Workflow DAG $W=(V,E)$, task requirements, cluster-size limit, and resource-capacity limits.

Output: Dependency-aware task clusters $D_1, D_2, \dots, D_K$.

1. Validate that W is acyclic and generate a topological order of its tasks.
 2. Construct the predecessor and successor sets for every task.
 3. For each candidate task pair, calculate successor correlation using (4.2).
 4. Calculate each task dependency strength DS_i using (4.3).
 5. Choose the unclustered task with the greatest dependency strength as a cluster seed.
 6. Add the most strongly correlated unclustered tasks while the cluster-size and resource-capacity constraints remain satisfied.
 7. Repeat Steps 5 and 6 until all tasks belong to exactly one cluster.
 8. Preserve the topological order inside every cluster and return the cluster set.
-

Algorithm 4. Dependency Aware PSO Scheduling

Input: Dependency clusters D ; VM set; swarm size P ; iteration limit R ; inertia w ; acceleration coefficients c_1 and c_2 .

Output: Best precedence-feasible workflow schedule S_{best} .

1. Initialize P particles as VM-assignment vectors with precedence-feasible priority keys.
 2. Decode each particle in topological order and compute execution, communication, dependency-delay, cost, and load metrics.
 3. Evaluate each schedule with (4.4) and the normalized objective in (4.6); penalize infeasible schedules.
 4. Initialize and update each personal best and the global best schedule.
 5. Update particle velocity using (4.5), then update and discretize its task-to-VM position.
 6. Repair precedence, release-time, and capacity violations before evaluating the new particle.
 7. Repeat Steps 2-6 until convergence or R iterations are reached.
 8. Return the global-best legal schedule S_{best} .
-

B Formal Analysis

Proposition 3. A topological decoder with communication-aware release times maintains every workflow edge.

Proof. For every edge (u, v) , u comes before v in topological order. The decoder does not start v before the maximum parent finish time plus the transfer delay, so the precedence constraint is satisfied. $\square$

Proposition 4. A repair operation that delays an invalid child cannot introduce a new incoming-edge violation.

Proof. The child's start time only increases. Descendants are then recomputed in topological order, and outgoing relationships are preserved (after a finite number of passes). $\square$

Pairwise dependency similarity has a worst-case complexity of $O(n^2 * d)$ where d is the bound on set intersection. Topological sorting takes $O(n \text{ plus the number of edges})$. For P particles, R iterations, and m machines, the worst-case search cost is $O(PR(nm \text{ plus the number of edges}))$. Dependency clusters can reduce the effective number of eligible machines and task units.

VI SCALABLE MULTI OBJECTIVE METAHEURISTIC SCHEDULING

A Proposed Model

SMMSF uses an iterative meta-heuristic optimizer because the fitness landscape is non-convex and multimodal. The optimizer starts with an initial population of candidate schedules, evaluates the fitness of each candidate, identifies the best solution, and then randomly moves the current solutions to a neighborhood. If a new schedule improves the fitness, it replaces the current one. The process continues until either the maximum number of iterations is reached or convergence is detected. This adaptive search allows the framework to avoid local optima and find high-quality mappings that a greedy or static heuristic might miss.

These three stages collectively establish a closed-loop scheduling pipeline: VM clustering mitigates search space size, multi-objective scheduling assesses candidate mappings, and meta-heuristic optimization iteratively refines the solution. The following three sections mathematically describe the algorithms.

Large-scale cloud scheduling approaches often optimize makespan, energy, or cost independently. Self-adaptive PSO enhances convergence, clustering-based ACO decreases mapping size, and resource-aware algorithms handle VM heterogeneity. Their isolated usage raises two concerns. But a single metric may push cost to another objective, and a population search over all task-to-VM pairs is too expensive at scale. SMMSF integrates observed VM profiles, multi-objective VM clustering, normalized fitness evaluation, and adaptive meta-heuristic refinement. This multi-level design limits the candidate space before optimization and evaluates each candidate over time, energy, cost, and utilization simultaneously.

The MOVMC algorithm takes a set of virtual machines $VM = \{vm_1, vm_2, \dots, vm_m\}$, each characterized by CPU, RAM, and bandwidth capacities, and a host-level power model. It also takes non-negative weights w_1, w_2, w_3 which determine the impact of each resource dimension on the VM capability index.

Initialization: If required, all VM parameters are normalized so that CPU, RAM, and bandwidth are similarly weighted in the capability score.

Capability Computation: For each VM vm_j , compute the scalar capability:

This score collapses multi-dimensional resource heterogeneity into a single comparable quantity.

Energy Estimation: We can estimate the energy consumption of each VM from its CPU utilization and host power:

Energy is added at the clustering phase, enabling the framework to create energy-efficient VM groups.

Similarity Measurement: The dissimilarity between any two VMs is the square of the difference of their capability scores:

for all intra-cluster VM pairs. This can be seen as a centroid-based partitioning problem which can be addressed by k-means or other similar clustering methods.

Output: After completion of the algorithm, a set of k VM clusters $C = \{C_1, C_2, \dots, C_k\}$ is obtained, where VMs of the same cluster are similar in terms of the capability and energy.

This algorithm takes as input a task set $T = \{t_1, t_2, \dots, t_n\}$, a set of VM clusters C yielded from the MOVMC, task parameters (task length, task deadline, task cost), VM parameters (MIPS, power, cost rate), and the weight factors α, β, γ and δ .

Initialization: Tasks and VM clusters.

Any preprocessing, such as sorting by deadlines or length normalization, has been done.

Run time Calculation: For every task t_i and virtual machine vm_j Has.

This is the core time-estimation formula used throughout the framework.

Makespan Computation: After assigning all tasks, the makespan is:

Makespan = $\max(\text{FinishTime}(t_i))$ for all $t_i \in T$

Energy Calculation: Dynamic power of a VM is linearly depending on its CPU utilization [30]:

$\text{Power}(vm_j) = \text{CPU_util}(vm_j) \times \text{Host_Power}$ Energy = Power $\times$ ET

Cost Calculation: The operational cost is given by the execution time and the cost rate of the VM:

$\text{Cost}(t_i) = \text{ET}(t_i, vm_j) \times \text{Cost_rate}(vm_j)$ --(47)

Utilization Computation: CPU utilization is computed as:

$U(vm_j) = \text{CPU_used}(vm_j) / \text{CPU_total}(vm_j)$ --(48)

Fitness Evaluation: The scalarized multi-objective fitness is:

$F = \alpha \cdot \text{Makespan} + \beta \cdot \text{Energy} + \gamma \cdot \text{Cost} + \delta \cdot (1 - U)$ --(49)

Evaluation of mappings: All the candidate task – V M mappings are evaluated by the above fitness function, and the fittest mappings are evolved by Algorithm 3 for further refinement.

Input: The optimizer receives an initial set of solutions S , the fitness function F , and a maximum number of iterations MaxIter .

Each solution $S \in S$ corresponds for a full mapping from task to VM.

Population Initialization: Create an initial population of candidate schedules, either at random or by using heuristic seeds from Algorithm 2.

a. Neighborhood Perturbation Generate a new candidate by perturbing the current solution:

Fitness Evaluation: Evaluate each candidate schedule using:

$F(S) = \alpha \cdot \text{Makespan} + \beta \cdot \text{Energy} + \gamma \cdot \text{Cost} + \delta \cdot (1 - U)$ --(50)

Best Solution Identification: Initialize the global best:

Iterative Improvement: Repeat until convergence or until MaxIter is reached:

a. Neighborhood Perturbation: Produce a new candidate by perturbing the current solution:

This random perturbation may be swapping two task assignments, shifting a task to another VM, or reassigning an entire task cluster.

Output: Return the best scheduling solution S^* discovered during the search.

The SMMSF framework is illustrated in Fig. 4 as a monitored, iterative scheduling cycle. It begins with the arrival of a large task set. Task-parameter estimation determines computation length, CPU and memory requirements, bandwidth demand, deadline, expected transfer size, and candidate execution cost. The values are normalized and retained at task level so that the optimization stage can evaluate time, energy, price, and utilization without allowing one metric scale to dominate the others.

Task analysis runs in parallel with the VM-analysis module. VM resource monitoring provides information about processor availability, RAM, bandwidth, queue state, utilization, power state, and current price. The VM capability calculation transforms the normalized resource vector into the capability value in Equation (5.1). Monitoring is done at the configured interval because a VM that was suitable at initialization may become overloaded, unavailable, or economically unattractive before dispatch. Thus, the derived capability is a time-indexed scheduling input, not a permanent machine label. The VM-clustering block combines machines of similar capability and current operating conditions. Clustering reduces the search space from all task-to-VM combinations to only compatible task-cluster to VM-cluster relations. Infeasible or saturated VMs are removed from the active candidate groups. This hierarchical reduction is the primary scalability mechanism of SMMSF: optimization is first done over relevant groups of resources and then extended to individual task assignments, thus avoiding evaluation of the entire mapping space.

The optimization module is composed of two connected operations. Multi-objective fitness evaluation involves the computation of normalized makespan, energy consumption, operational cost, and utilization loss using (5.2)-(5.6). Meta-heuristic optimization then searches for a feasible mapping with the smallest combined fitness, keeps strong yet diverse solutions, and fixes capacity or deadline violations. The objective coefficients represent the preference of the experiment, and normalization is the key to combining milliseconds, kilowatt-hours, cost units, and percentages in a consistent manner. New capability and fitness evaluation is triggered by changes identified through VM monitoring.

The chosen solution is transformed into task-VM mappings and sent to the load-balancing decision. The tasks are released for execution if the utilization spread is acceptable (the "Yes" branch). In the case that it is not acceptable, the "No" branch modifies the mapping and sends it back to the optimization path. The re-optimization is restricted to the affected assignments so that the online adaptation does not need to restart the in-depth search from scratch. The framework also provides a report of the makespan, energy, cost, and utilization after the execution of the tasks. Figure 5.1 thus represents a closed loop scheduler where monitoring, clustering, optimization, mapping, and load testing are iterative until a feasible balanced schedule is found.

$$Cap_j = a CPU_j + b RAM_j + c BW_j \quad (5.1)$$

$$M(S) = \max_i FT_i \quad (5.2)$$

$$E(S) = \sum_j \int P_j[U_j(t)] dt \quad (5.3)$$

$$C(S) = \sum_i p_{rice}(s_i, vm_j) \quad (5.4)$$

$$U(S) = \frac{\sum_j busy_j}{\sum_j available_j} \quad (5.5)$$

$$F(S) = aM_n + bE_n + cC_n + d(1 - U_n) \quad (5.6)$$

$$a + b + c + d = 1$$

$$f_n(S) = \frac{f(S) - f_{min}}{f_{max} - f_{min}} \quad (5.7)$$

Equation (5.1) maps CPU, RAM, and bandwidth to a unified capability value of VM j . The coefficients represent the resource pattern that the current workload predicts to use. The CPU factor can be modified for tasks that are known to be heavily CPU bound, and the bandwidth factor can be changed for data heavy workflows. This score is not used for replacing hard capacity constraints. It is used as similarity coordinate during MOVMC, but at the end of the day, each selected assignment is tested against residual capacity.

The makespan in (5.2) is the maximum finishing time that corresponds to the execution of last scheduled task. The energy (5.3) is the integral of the VM power (5.1) with respect to time. A standard linear model linearly interpolates between the idle and peak using the utilization as a factor, although nonlinear server-specific models can be used. Equation (5.4) adds up the monetary cost charging for the execution time and VM cost. These three numbers are in different units and scales, so the raw values can not be directly added.

(5.5) is only the fraction of the available VM time spent by a productive VM at the execution. The $1-U$ complement is present in (5.6) as well, thus every term of the function has been minimized. Normalized quantities n calculating with respect to the observed or policy defined ranges are represented as subscript n . a , b , c and d are the coefficients which add up to one that indicate the current provider preference. A sensitivity experiment should also be conducted in which these parameters are varied in a systematic manner, the ensuing trade-offs recorded, and an assessment established of the extent to which the reported schedules hold up to moderate changes in preferences.

The SMMSF experiments use 3 datacenters, 200 hosts, 500-1000 heterogeneous VMs, and workloads of 5000, 7500, and 10000 tasks. Hosts provide 64 GB RAM and 4 TB storage; VMs provide 4-16 GB RAM, 1 Gbps bandwidth, and 1000-10000 MIPS. The optimizer uses a population of 60, 150 iterations, and 30 runs. The normalized weights are $a=0.35$ for makespan, $b=0.30$ for energy, $c=0.20$ for cost, and $d=0.15$ for utilization loss. Normalization prevents any raw metric scale from dominating the combined fitness.

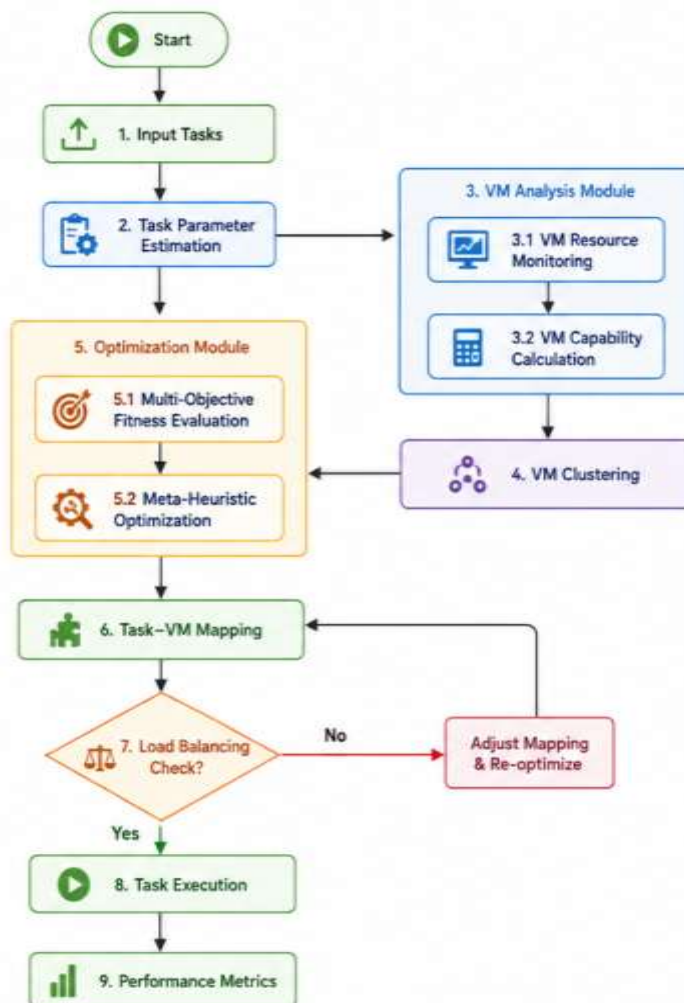


Fig. 4. SMMSF closed-loop scheduling framework.

Algorithm 5. Multi Objective VM Clustering

Input: VM set with CPU, RAM, bandwidth, utilization, power, and price attributes; cluster count K ; weights a , b , and
Output: VM clusters $C_1, C_2, \dots, C_K$ with comparable capability and energy profiles.

1. Normalize CPU, RAM, and bandwidth attributes.
2. For every VM, calculate the capability score Cap_j using (5.1).
3. Estimate the current power or energy characteristic from VM utilization and the host power model.
4. Construct a capability-energy feature vector and initialize K cluster centroids.
5. Assign each VM to the nearest compatible centroid and recompute the centroids.
6. Repeat Step 5 until assignments stabilize or the iteration limit is reached.
7. Remove unavailable or saturated VMs and return the resulting VM clusters.

Algorithm 6. Scalable Multi Objective Task Scheduling

Input: Task set T ; VM clusters C ; task and VM parameters; objective weights a , b , c , and d .

Output: A feasible population of high-quality task-to-VM schedules.

1. Normalize task attributes and order ready tasks by deadline, priority, and dependency status.
2. Identify compatible VM clusters for each task and discard capacity-infeasible candidates.
3. For each feasible task-VM pair, compute execution time and predicted finish time.
4. Construct candidate mappings from the compatible clusters and enforce one assignment per task.
5. For every mapping, compute makespan, energy, cost, and utilization using (5.2)-(5.5).
6. Normalize the objectives using (5.7) and evaluate combined fitness using (5.6).
7. Retain the best feasible and diverse mappings for adaptive refinement.

Algorithm 7. Adaptive Metaheuristic Refinement

Input: Candidate schedule set S ; fitness function F ; maximum iterations $MaxIter$; convergence tolerance ϵ .

Output: Best refined schedule S_{star} .

1. Initialize the population from the candidate mappings and evaluate every schedule.
 2. Set S_{star} to the feasible schedule with the lowest fitness.
 3. Generate a neighboring candidate by swapping two assignments, moving one task, or reallocating one task cluster.
 4. Repair capacity, deadline, and dependency violations in the candidate.
 5. Evaluate the repaired candidate with (5.6) and accept it when it improves its incumbent schedule.
 6. Update S_{star} whenever a lower-fitness feasible schedule is found.
 7. Repeat Steps 3-6 until $MaxIter$ is reached or the improvement is below ϵ .
 8. Return S_{star} .
-

B Formal Analysis

Proposition 5. If all objectives are normalized between zero and one, and weights are nonnegative and sum up to one, the fitness is normalized between zero and one.

Proof. Utilization loss is also in the unit interval. The fitness is a convex combination so it can't leave the range of its components. $\square$

Proposition 6. A finite population size, finite iteration bound, and bounded repair ensure termination.

Proof. Finite estimation, update, and repair at each iteration. The counter stops no later than the specified limit, while elitism retains the best feasible schedule found so far. $\square$

VM clustering takes $O(rmKq)$ for r iterations with q features and m q -dimensional vectors. Refinement of the population takes $O(PRnm\text{-prime})$ when $m\text{-prime}$ machines are still compatible, i.e. Following monitoring adds $O(Hm)$ for H refresh events. Hierarchical reduction is relevant only if $m\text{-prime}$ is significantly smaller than m the; the trivial case returns the exhaustive search bound.

VII EXPERIMENTAL METHODOLOGY

CloudSim 6.0 simulates QTSF and SMMSF, whereas WorkflowSim simulates workflow DAGs for DAWSM. QTSF uses 30 runs with random seed 50, and SMMSF uses 30 runs. The available results are aggregated rather than run-level

observations; therefore, the reported comparisons are limited to arithmetic and relative improvements, without unsupported standard deviations, confidence intervals, p-values, or effect sizes.

TABLE II. Simulation Settings

Parameter	QTSF	SMMSF
Datacenters	1	3
Hosts	50	200
VMs	100-500	500-1000
Tasks	500-5000	5000 7500 10000
VM MIPS	1000-5000	1000-10000
VM RAM	2-8 GB	4-16 GB
Runs	30	30
Iterations	100	150

For clusters of tasks, if the algorithm runs for iterations, the complexity is roughly for the QoS clustering, plus an identical term for resource refinement. This is polynomial and thus scalable to large sets of tasks, unlike the exponential search space of the NP-hard scheduling problem it solves.

However, a key insight is that the algorithm employs a linear, additive QoS model. Although this makes it easier to implement and understand, it assumes that task length, priority and deadline are independent of each other and are compensatory (i.e., you can trade off one for another).

The second part of the algorithm is ClusterMapper, which assigns the clusters of tasks generated by Algorithm 1 to the available VM infrastructure. It is specified in the paper as follows:

In Step 3, the VM capacity is defined as the sum of CPU, RAM and bandwidth capacities. This one-dimensional representation allows for a simple one-dimensional best-fit search in step 4. The assignment rule looks for the VM that has the total capacity closest to the task's total requirement. This is a bin-packing heuristic that tends to pack tasks tightly, leaving no large residuals on any single VM.

Step 5 then recalculates the actual execution time using the MIPS rating of the selected VM after assignment. Step 6 tests the global load for balance by the minimum and maximum utilizations of the VMs. If the difference between the VM with the maximum load and that with the minimum load is larger than a predefined threshold, step 7 will execute a series of task migrations from heavily loaded VMs to lightly loaded VMs. This feedback mechanism is what differentiates QTSF from static best-fit schedulers and puts it in line with dynamic load-balancing goals.

The QTSF configuration reports all optimization parameters required for reproducibility. QTSF first clusters tasks using QoS and resource attributes and then assigns them to capability-compatible virtual machines. Scheduling is consequently guided by multiple QoS measures rather than a single criterion.

At first, a cloud task has been modeled with a set of different scheduling parameters such as task length, execution priority, deadline, required amount of CPU, memory, and bandwidth. These features are normalized to remove the effect of scale difference prior to calculate the Quality of Service score.

The Quality of Service score is calculated using

The following value of parameters have been used for the simulation after several trial of simulations.

The reason for the higher weight for length is that it has a more direct impact on the scheduling efficiency, while priority and deadline together ensure the QoS. the scheduler reassigns the tasks from heavily loaded virtual machines to lightly loaded virtual machines dynamically.

Task similarity during clustering is computed using

Load balancing is continuously monitored using

Whenever the load balancing threshold exceeds the scheduler dynamically reallocates tasks from overloaded virtual machines to lightly loaded virtual machines. The optimization process is stopped at any of the following stopping conditions. These optimization parameters are integrated into the QTSF methodology to support reproducibility and independent experimental validation. Actually, VM selection is based on a multi-parameter scheduling model, taking into consideration QoS, resource compatibility, workload balancing, execution time estimation, as well as VM availability. Task priority is only one part of the scheduling decision. Each task is represented by a number of parameters such as task length, execution priority, deadline, CPU requirement, memory requirement, bandwidth requirement, and estimated execution time during QoS estimation). These factors all contribute to priority scheduling. Following QoS evaluation, the scheduler computes the resource requirement of every task using Similarly, each virtual machine capability is evaluated as The scheduler assigns tasks according to the minimum capability mismatch Such a resource-conscious mapping can avoid a big powerful virtual machine being underutilized by small tasks, while a small virtual machine being overloaded. The scheduler further estimates execution time to find the estimated completion time of each task prior to allocation. If adequate computational facilities are accessible, the tasks with shorter execution time and more urgent deadline may also be scheduled first. Load balancing is continuously monitored through VM utilization is updated periodically. When the difference of the utilizations exceeds the defined threshold, tasks are migrated dynamically from an overloaded virtual machine to an underloaded one. Hence, the virtual machine assignment not only relies on task priority, but a few other scheduling parameters. The principal scheduling parameters are: Taking into account more parameters allows the proposed framework to produce a lower makespan, higher resource utilization, a balanced workload, and better QoS than traditional priority based scheduling schemes. These thresholds were determined by a cut and try method based on the result of extensive simulation runs in order to achieve trade-off among QoS, execution efficiency, and resource utilization. Higher-priority tasks are scheduled earlier than lower-priority ones in the workload. But priority is not enough to make the decision of scheduling. Consider virtual machine capacity and resource utilization, load balancing and resource compatibility, and then complete task allocation. This multi-parameter priority estimation mechanism makes the proposed scheduler to reduce makespan, enhance resource utilization and deadline satisfaction, and considerate balanced cloud resource usage when compared with existing priority-based scheduling approaches scheme in terms of QoS expectation. The hardware platform and software for the experiments are given in the revised abstract to enhance the reproducibility of the present work. Because the Quality of Service-Aware Task Scheduling Framework (QTSF), Dependency-Aware Workflow Scheduling Model (DAWSM), and Scalable Multi-Objective Meta-Heuristic Scheduling Framework (SMMSF) are scheduling algorithms based on simulation, the realization was performed in the CloudSim and WorkflowSim simulators. CloudSim 6.0 was selected because it models datacenters, virtual machines, cloudlets, brokers, hosts, and scheduling policies without requiring physical cloud infrastructure. WorkflowSim, an extension of CloudSim, was used for DAWSM to simulate workflow execution, task dependencies, directed acyclic graph scheduling, and communication delay. The algorithm is implemented in Java using Eclipse Integrated Development Environment (IDE). Implementing the Java Development Kit (JDK 17) CloudSim naturally is entirely Java-based and does so offer simple to complex algorithms for your needs. Experiments were performed on a workstation with Intel Core i7 Processor (12th Generation), 16 GB DDR4 RAM, 512 GB SSD storage, Windows 11 (64-bit), JRE Version 17. The workstation provided ample power to simulate thousands of cloud tasks and heterogeneous virtual machines without any computational bottlenecks. The simulation architecture consists of the following software components:

The aforementioned scheduling algorithms were embedded in a DatacenterBroker module within CloudSim. Task generation, clustering, VM creation, scheduling decisions, as well as monitoring of performances, are separated Java modules, in order to facilitate experimentation and/or future extensions.

There are six major steps involved in the processing flow of implementation. User tasks Generation at first, then submission to the CloudSim broker. The scheduler calculates QoS scores, clusters the tasks, approximates the capabilities of virtual machines, and runs the suggested optimization algorithms. At last, CloudSim logs execution statistics such as makespan, execution time, throughput, energy, resource utilization, and computational cost.

The implementation environment supports controlled and repeatable experiments under multiple workload conditions. All implementation parameters are explicitly reported to enable independent verification of the scheduling methods.

The CloudSim configuration and simulation settings are specified explicitly. The scheduling schemes were evaluated with CloudSim 6.0 in a heterogeneous cloud environment containing multiple datacenters, hosts, VMs, and cloudlets.

The simulation environment was set to represent the IaaS cloud model based on heterogeneous resources. To test the scalability of the designed scheduling algorithms, a number of workload sizes were created.

The CloudSim configuration adopted during experimentation is summarized below.

The experiments were performed under three workload categories.

Since DAWSM is aimed at workflow scheduling, the simulation environment should be able to handle directed acyclic graphs and dependency-aware execution. The simulation results presented in this objective were obtained using Workflow Sim, a CloudSim toolkit extension that natively supports workflow modeling, task dependencies, and data-transfer delays. The simulated cloud data center is composed of heterogeneous VMs with different MIPS ratings, RAM sizes, bandwidths, and cost rates. We evaluated dependency satisfaction for various graph structures and task sizes using workflow benchmarks like Montage, Cyber shake, Epigenomics, and In spiral or their functionally equivalent synthetic DAGs. Various task sizes, from small computational kernels to large data-intensive jobs, and various VM counts were used to test load distribution.

We compared the proposed DAWSM with three baseline approaches: PSO+DBCA, ACO+DBCA, and Alpha+DBCA.

The performance metrics that were recorded were:

Dependency Satisfaction (%) — fraction of precedence constraints honored,

Execution Time (ms) — total workflow completion time,

Clustering Efficiency (%) — quality of the dependency-based clusters, and

Computational Cost — cumulative cost of VM usage during execution.

Dependency satisfaction. DAWSM achieves 95% dependency satisfaction, which is higher than PSO+DBCA (88%), ACO+DBCA (85%), and Alpha+DBCA (80%). This enhancement is a direct consequence of the explicit dependency constraint handling in DAS-PSO. In contrast to the baseline algorithms that consider the dependency preservation as a second-order priority, DAWSM makes it into the search process.

Running time . The proposed approach finishes the procedure in 4200 ms, 16 % faster than PSO+DBCA, 20.8 % faster than ACO+DBCA, 27.6 % faster than Alpha+DBCA. The reduction is due to DBTC's successor-aware clustering: DAWSM reduces the inter-VM data transfers and running idle waiting time by clustering together tasks that have common successors.

Efficiency of clustering. DAWSM achieves a clustering efficiency of 92%, which is better than the baselines by a good margin, i.e. 84%, 80%, 75%. The higher value means that DBTC generates more cohesive workflow clusters, which facilitate scheduling and locality.

Computing cost. The computational cost for our model is 225 units compared to 300, 340, and 400 for the baselines. This correspond to savings of 25%, 33., and 43.75%, in order. Even with the additional optimization burden of PSO, the improved clustering and better VM usage translate into a smaller overall cost.

Important observations. While DAWSM demonstrates steady progress, the satisfaction of the dependencies being 95% high also suggests that a small portion of constraints might still be violated and re-processed instead of being fully satisfied. An exact / hybrid approaches could be explored to increase this measure towards 100% as a future work. Furthermore, the evaluation is simulated; validation on actual cloud traces (e.g., Azure, AWS) would provide more practical evidence.

The DAWSM evaluation metrics are defined as follows. E is the workflow-edge set, $I(.)$ is an indicator that a precedence relation is satisfied, CT_i is task completion time, and inter-cluster edges represent dependency cuts between clusters.

WorkflowSim scenarios are constructed based on Montage, CyberShake, Epigenomics and Inspiral DAGs or synthetic DAGs with equivalent size. The implementation evidence via the workflow parser, the DAG structure, the ready-task queue, the particle-fitness trace, the repaired precedence order, and the end WorkflowSim output, each bestows a well-defined phase of execution and established linkages among sections of the produced artifact. DAWSM is compared with PSO+DBCA, ACO+DBCA, and Alpha+DBCA, representing swarm, ant-colony, and dependency-clustering baselines. The baseline choice and interpretation are supported by workflow scheduling and composite-task PSO studies [19, 21, 29, 30, 31, 33]. DAWSM gives higher dependency satisfaction and clustering efficiency with lower execution time and computational cost.

1. The heterogeneous data center, which is a good representative of Experimental Setup a real system, consisted of multiple hosts with different CPU capacity, memory and power models. Virtual machines with varying MIPS values (usage levels), RAM sizes (working data), bandwidth capacity (working data), and cost rates (market prices) are considered to mimic real cloud offerings. The weights α , β , γ , and δ of the fitness function were tuned to have a similar numerical value in the four objectives, so that no an objective could dominate the combined fitness function. The baselines chosen for comparison were SRA, SOPSO, and CLUSTERACO, which correspond to scalarized ranking, particle swarm optimization, and ant colony clustering methods, respectively. Each experiment was conducted multiple times on different workload instances and random seeds to guarantee the statistical validity.

The experiments were carried out using CloudSim Version 6.0 at IaaS layer. Large scale workloads were simulated to test the scalability and reliability of the scheduling framework in cloud environment. Table 4 summarizes the simulation parameters used in the experiment.

SMMSF algorithm to optimize four scheduling objectives at the same time. The fitness function is defined as:

The total energy consumption is calculated using

The adaptive optimization algorithm uses the following search parameters.

Three workload categories were considered during experimentation.

The proposed approach should also be compared against SRA, SOPSO and CLUSTERACO under the same simulation settings.” We have considered makespan, execution time, energy consumption, operational cost, throughput, load balancing, and resource utilization as the performance metrics.

Experimental results indicate that the proposed SMMSF achieves the consistently lowest makespan, least energy consumption, least operational cost, best resource utilization, and best load balancing among all the schemes under various workload conditions. The adaptive resource monitoring scheme allows the scheduler to adapt dynamically to changes of the workload, and hence the proposed model is well suited for real large-scale cloud computing environment.

The SMMSF performance was measured by using the following four main metrics: makespan (ms), energy consumption (kWh), operational cost (units), and resource utilization (%). The comparative outcomes documented in the review paper are summarized in Table 1.

The results show that SMMSF achieves better performance than all baselines for all the metrics. The makespan of 4100 ms brings about 14.6%, 18.0%, and 26.8% of improvements over the ones achieved by SRA, SOPSO, and CLUSTERACO, respectively. Such reductions are due to the ability of the framework to select suitable VM capabilities for task computational requirements and to balance the load among clusters.

The energy consumption is decreased to 210 kWh, which is 19.2%, 25.0%, and 34.4% less than that of SRA, SOPSO, and CLUSTERACO, respectively. With energy-aware VM clustering and power-based fitness evaluation, the SMMSF can also prevent high power consuming VM assignment if it has lower-energy consuming alternatives.

Operational costs have the same pattern. At 120 units, the SMMSF is able to save 20.0%, 27.3%, 40.0%, against SRA, SOPSO and CLUSTERACO, respectively. The cost-related fitness term makes sure that costly high-performance VMs are utilized only if these are warranted by respective reductions in makespan or energy.

In particular, the resource utilization of SMMSF reaches the best performance with ‘94%’, which is much higher than 85%, 82% and 70% of baselines. This high utilization suggests that the framework achieves a very low resource tended to be in idle state and workload consolidation is also attained, which is a critical factor for cloud provider earnings.

These improvements are the result of three factors working in concert. First, VM clustering simplifies the search space by clustering at-rable resources. Second, the m-objective fitness function guarantees that the decisions of the schedules are not measured using a single measure. Thirdly, since the meta-heuristic algorithm iteratively improves the solution adaptively, it allows also to escape from local optima which are the solutions that simpler

heuristics get stuck. Thus, a limitation is that the current scalarized fitness requires manual weight tuning; future work could include studying how weights can be adjusted adaptively or how to generate the Pareto-front explicitly.

The medium, large and the very large workload scales (see Table 5.1) used in the large-scale CLOUDSIM set-up together with the following infrastructure parameters: Simulation and Execution Environment.

SMMSF uses the makespan, energy, cost, and utilization measures in (5.2)-(5.5). Each metric is normalized using (5.7), and the combined schedule fitness is calculated using (5.6).

Large-scale implementation views present VM-monitor updates, MOVMC cluster membership, population fitness by iteration, feasibility-repair events, and final CloudSim summaries for 5000, 7500, and 10000 tasks. Each view corresponds to a reproducible stage of the SMMSF execution sequence.

SMMSF is compared against SRA, SOPSO, and CLUSTERACO as representative scalar ranking, self-adaptive particle-swarm, and clustering-ant-colony schedulers. The comparison is made with multi-objective genetic, hybrid ant-genetic, enhanced meta-heuristic, and scheduling-review papers [18, 22, 32, 34, 35, 36, 37, 38]. The results are in favor of SMMSF for all four normalized objectives.

VIII RESULTS

A QTSF Results

TABLE III. QTSF Results

Metric	QTSF	K-Means+ACO	PSO	Traditional
Makespan ms	5693	6233	6793	6933
Load balance %	92	78	74	65
Utilization %	88	76	73	60

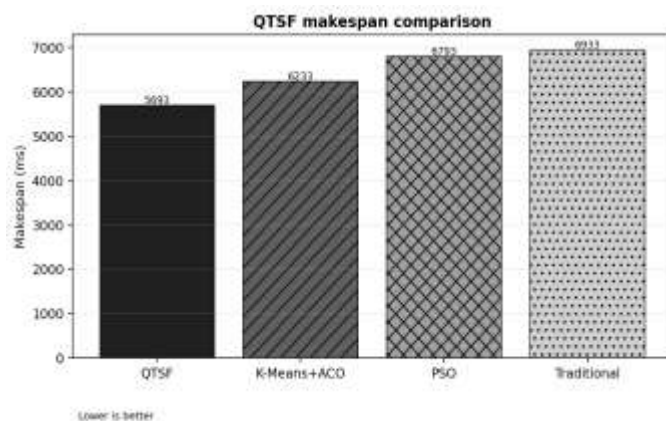


Fig. QTSF makespan.

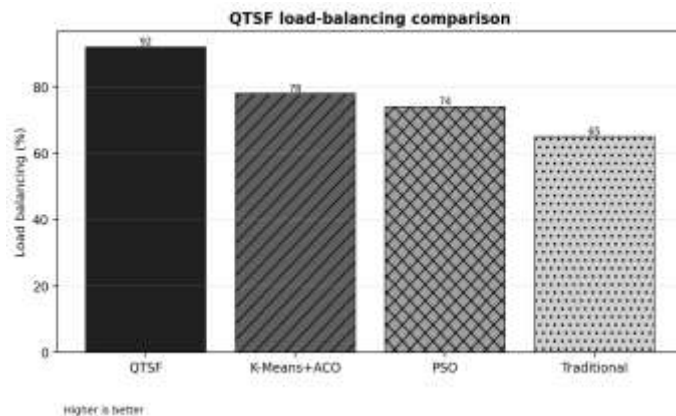


Fig. QTSF load balancing.

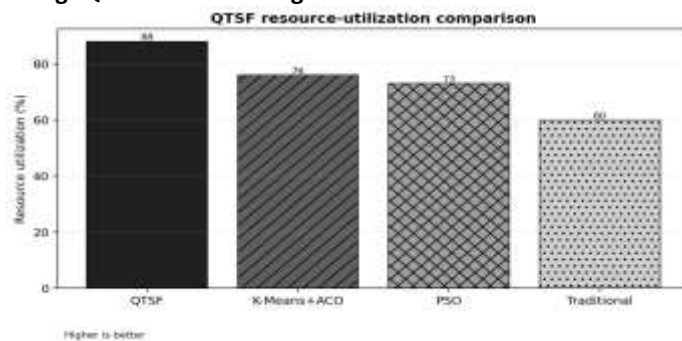


Fig. QTSF resource utilization.

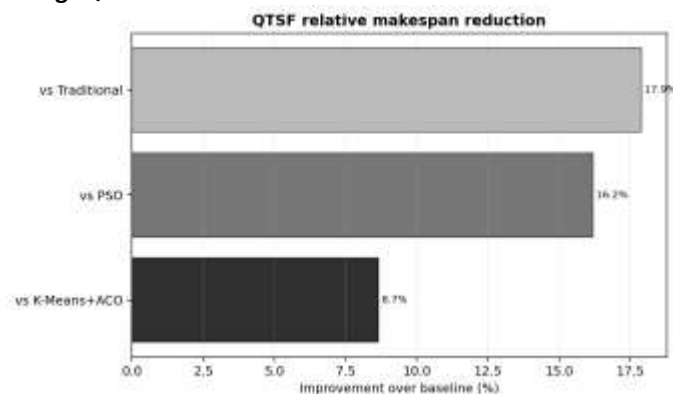


Fig. QTSF relative makespan reduction.

The Experimental Design consists of 500–1000 tasks for small workloads (SWL), 1000–3000 tasks for medium workloads (MWL), and 3000–5000 tasks for large workloads (LWL). Each scenario is run 30 times. Runtime logs broker initialization, task-cluster formation, task-to-VM assignments, completion events, utilization samples, and final metric values. Les parcelles expérimentales et les vues d'exécution qui sont rapportées semblent correspondre à ces étapes d'exécution.

QTSF performance is measured using makespan, load-balance efficiency, and resource utilization. FT_i is the finish time of task i , U_j is the utilization of VM j , m is the number of VMs, $busy_j$ is the productive time of VM j , and $available_j$ is its available time.

QTSF is benchmarked against the clustering+ACO, PSO scheduling and heuristic methods over the research of cloud-scheduling that involves clustering, swarm optimization, meta-heuristic techniques [20, 26, 27, 30, 32]. Under the aforementioned setup, QTSF achieves the best makespan, load-balancing, and resource utilization.

FIGURE 3.3 turns makespan on its side so that the time benefits are not visually masked by percentage figures. QTSF obtains 5693 ms, K-Means+ACO 6233 ms, PSO 6793 ms, and traditional scheduling 6933 ms. The absolute

reductions are 540, 1100, and 1240 ms. In terms of percentage, QTSF achieves an average reduction in the makespan of about 8.7%, 16.2%, and 17.9%. This echoes the claim that clustering on its own is not enough: the additional QoS ranking, capability matching, and load-repair phases make for better (shorter) completion time. Figure 3. 4 shows a 92% load-balancing value for QTSF. This is 14 percentage points above K-Means+ACO, 18 points above PSO, and 27 points above traditional scheduling. This finding is consistent with the explicit dispersion test in Algorithm 3.2. The current utilization range is checked after each tentative assignment; if the imbalance threshold is exceeded, an eligible task is moved. This feedback stage prevents a few fast VMs from taking all the queue, as the plot shows.

QTSF achieves 88% resource utilization, as shown in Figure 3.5. 76%, 73%, and 60% are the respective baseline values. Higher utilization does not mean indiscriminate saturation. QTSF places tasks on appropriate machines, idle capacity is reduced, and overload correction is still enabled. The simultaneous improvement in makespan and utilization is important: it shows that the utilization increase comes from performing actual work, not from longer queues or delayed completion.

In Figure 3.6, the raw makes pan values are converted into relative reductions. The K-Means+ACO to traditional baseline advantage is growing, which shows that each missing model component incurs a measurable penalty. These are deterministic transformations of Table 3. 2 and should not be considered as independent trials. Confidence intervals Should be added and simulation runs should be repeated when raw run-level observations are available.

The QTSF results provide a clear indication that QoS scoring, resource compatible clustering, capability matching and load repair constitute integrated phases. Results report the lower makespan for the proposed approach with higher load balance and utilization. The evidence is limited to the provided CloudSim configuration and aggregate result values; repeated trials, parameter sensitivity analysis, and production cloud validation are the principal next steps.

QTSF reduces the makespan by 8.7, 16.2, and 17.9 percent compared to K-Means+ACO, PSO, and the conventional scheduling. Load balance increases by 14, 18 and 27 percentage points while utilization increases by 12, 15 and 28 points. The joint configuration is consistent with service clustering, capability matching and load repair.

B DAWSM Results

TABLE IV. DAWSM Results

Metric	DAWSM	PSO+DBCA	ACO+DBCA	Alpha+DBCA
Dependency %	95	88	84	78
Execution ms	4200	5000	5300	5800
Clustering %	92	84	80	74
Cost units	225	270	295	330

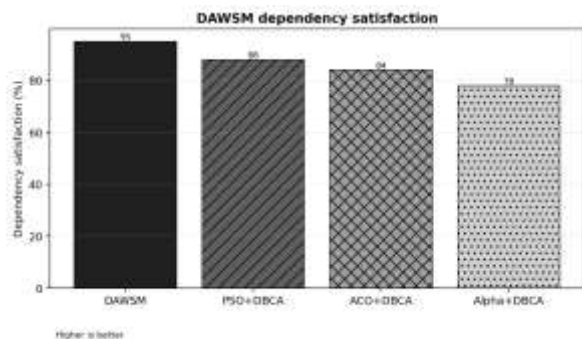


Fig. DAWSM dependency satisfaction.

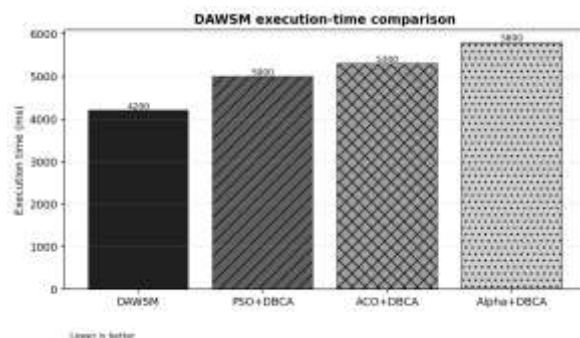


Fig. DAWSM execution time.

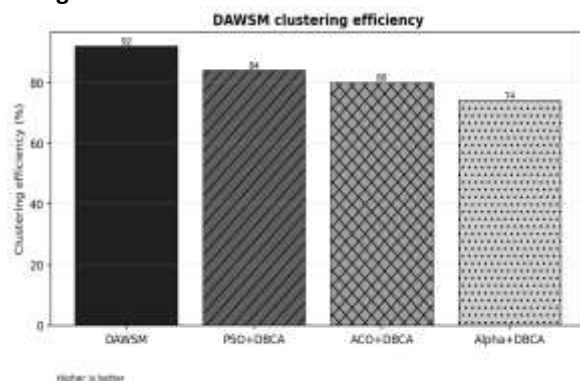


Fig. DAWSM clustering efficiency.

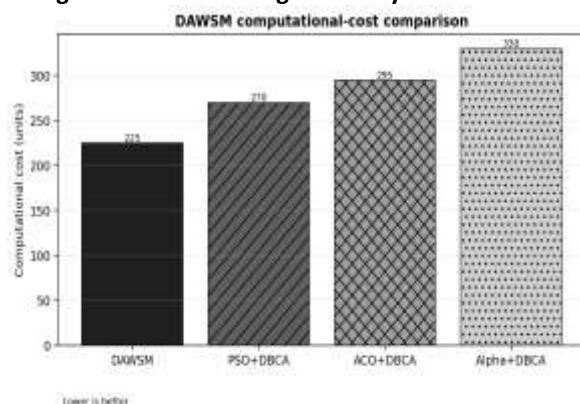


Fig. DAWSM computational cost.

The DBTC algorithm is the first algorithmic block of DAWSM. The aim is to break down a workflow DAG into clusters of tasks that are strongly connected by dependency relations, thereby reducing the downstream scheduling problem and enhancing data locality.

Description of important stages. Step 2: The raw workflow DAG is transformed to a local dependency structure for each task. Step 3 quantifies the similarity between two tasks in terms of their downstream dependencies. A large overlap between two tasks means that they drive similar subsequent computations and should thus be grouped together. Step 4 combines these pairwise similarities into a dependency score for each task, and Step 5 clusters the tasks by maximizing the sum of dependency scores over all clusters.

DBTC is dominated by the computational complexity of Step 3, which involves pairwise comparisons between tasks (in the worst case) for a workflow with tasks. For moderately sized scientific workflows, this is acceptable, and can be sped up with graph neighborhood sampling for very large DAGs. The output clusters are then fed to DAS-PSO as input, where they help to formulate feasible schedules with low latency.

The second algorithmic block of DAWSM is DAS-PSO. It takes the clustered workflow produced by DBTC and searches for a high-quality, dependency-feasible mapping to VMs.

Explanation of key steps. In Step 1, each particle encodes a candidate schedule, typically as a vector of VM assignments for the clustered tasks. Step 2 defines the optimization objective as a weighted sum of total execution time and dependency delay, where w_1 and w_2 are user-defined weights reflecting the relative importance of speed versus precedence satisfaction.

Steps 4 and 5 are where the PSO exploration mechanism is implemented. The inertia weight is a parameter that controls the momentum of the particle, while the cognitive coefficient and social coefficient represent the pull of the particle towards its personal best and the swarm's global best, respectively. Random factors ensure stochastic exploration.

Step 6 is the most crucial step since that is where the dependency constraints are enforced. For each directed edge the algorithm checks that does not precedes . If one particle does not satisfy this constraint, the schedule maker repairs the schedule by shifting the offending job, or it penalizes the fitness of the schedule. This method of dealing with constraints is emphasized in the review paper as a innovative aspect of the model.

When the swarm converges or the maximum number of iterations is reached, the algorithm terminates. The final solution corresponds to the best dependency-aware workflow schedule discovered.

In contrast to traditional workflow schedulers, which only focus on execution time optimization, the proposed Dependency-Aware Scheduling with Particle Swarm Optimization (DAS-PSO) considers various workflow features to assess each potential scheduling solution. Each particle in the swarm corresponds to a complete task-to-virtual-machine assignment. An individual particle's fitness is evaluated by a multi-objective fitness function.

The values adopted during experimentation are

where the maximum completion time determines the workflow makespan.

Dependency delay denotes the waiting time caused by precedence constraints and can be computed as:

Only positive waiting intervals are considered.

The Particle Swarm Optimization algorithm updates candidate schedules using

The optimization ends when the heat of one or more of the following stop criteria is reached:

No improvement in global fitness during ten consecutive iterations

The new fitness function, which evaluates at the dependency level, minimizes execution delay, delay due to dependency, computational cost, and workload balancing at the same time, leading to dependency-aware schedules with better execution-time efficiency.

As DAWSM is intended for workflow scheduling, the simulation environment needs to have support for directed acyclic graphs (DAG) and dependency-aware execution. The experiments reported in this goal are carried out by using Workflow Sim, a CloudSim based framework extending the latter with native support for modeling workflows, task dependencies, data-transfer delays. We assume heterogeneous VMs in the simulated cloud data center with different MIPS ratings, RAM sizes, bandwidths and cost rates. Workflow benchmarks such as Montage, Cyber shake, Epigenomics and In spiral (or ... functionally equivalent synthetic DAGs) were used for the study of dependency for different graph structures and task size through different Baseline #2: CloudMF The experiments reported in this objective were conducted using Workflow Sim (see below), an extension of the CloudSim toolkit that provides native support for model workflows, task dependencies, and data-transfer delays. Tasks ranged in size from small computational kernels to large, data-intensive job, and VM counts were scaled to experiment load distribution.

The proposed DAWSM was compared against three baseline methods: PSO+DBCA, ACO+DBCA, and Alpha+DBCA.

The performance metrics recorded were:

Dependency Satisfaction (%) — fraction of precedence constraints honored,

Execution Time (ms) — total workflow completion time,

Clustering Efficiency (%) — quality of the dependency-based clusters, and

Computational Cost — cumulative cost of VM usage during execution.

Dependency satisfaction. DAWSM achieves 95%dependency satisfaction, higher than PSO+DBCA(88%), ACO+DBCA(85%), Alpha+DBCA(80%) and others. Such an improvement originates directly from the explicit dependency constraint handling in DAS-PSO. While the baseline approaches consider dependency preservation as a post-processing matter, DAWSM incorporates it into the search.

Execution time. The proposed approach terminates the workflow in 4200 ms, which is 16%, 20.8% and 27.6% faster than that of PSO+DBCA, ACO+DBCA and Alpha+DBCA, respectively. The reduction can be attributed to DBTC's successor-aware clustering: by clustering such tasks that have common downstream tasks, DAWSM also minimizes inter-VM data transmissions and waiting idle time.

Clustering efficiency. DAWSM reaches 92% clustering efficiency, compared to 84%, 80% and 75% for the baselines. The greater value indicates that DBTC generates more compact workflow clusters, and the increased locality is beneficial to the schedule.

Computational cost. The computational cost of the proposed methods is 225 units, while that for the baselines is 300, 340 and 400, respectively. This corresponds to a saving of 25%, 33.% and 43.75%, respectively. Even with the added optimization overhead in PSO, the enhanced clustering and VM utilization efficiency allow for the reduction in the total cost.

Critical observations. Although DAWSM exhibits consistent performance increases, the 95% dependency satisfaction — albeit high — shows that there still might exist a small portion of constraints that can be handled by penalty or repair rather than absolute satisfaction. Future work might consider exact or hybrid approaches to drive this metric to 100%. Furthermore, evaluation is done based on simulation, which if validated with real cloud traces (e.g., Azure, AWS) will be able to enhance the practical claims.

The DAWSM metrics use E for the workflow-edge set, $I(.)$ as an indicator of precedence satisfaction, CT_i as task completion time, and inter-cluster edges as dependency cuts.

WorkflowSim scenarios employ Montage, CyberShake, Epigenomics, and Inspiral DAG templates, or size-equivalent synthetic DAGs. The Implementation evidence demonstrates the workflow parser, DAG, ready-task queue, particle-fitness trace, repaired precedence order and the final WorkflowSim output, where each view is connected to an specific execution stage.

DAWSM is compared to the swarm, ant-colony, and dependency-clustering baselines: PSO+DBCA, ACO+DBCA, and Alpha+DBCA. The choice of baseline and its interpretation are corroborated by task workflow scheduling and composite-task PSO results [19,21,29,30,31,33]. DAWSM provides better dependency satisfaction and clustering efficiency at lower execution time and computational cost.

DAWSM achieves 95% dependency satisfaction, which is 7 percentage points higher than PSO+DBCA, 11 points higher than ACO+DBCA, and 17 points higher than Alpha+DBCA. This difference comes from two controls: DBTC groups tasks using shared dependency structure, and the DAS-PSO repair stage rejects or corrects candidate schedules that violate parent-child ordering. The remaining 5% of the gap should be explored with failure analysis specific to the workflows, including communication delay, cluster boundaries, and how complex join nodes are handled.

Figure 4.4 shows DAWSM at 4200 ms, a 16.0%, 20.8%, and 27.6% reduction compared to PSO+DBCA, ACO+DBCA, and Alpha+DBCA respectively. The progressive difference correlates with less waiting at dependency boundaries and more effective VM matching. This result must be considered in conjunction with Figure 4.3: the faster completion is achieved without compromising precedence satisfaction.

DAWSM cluster efficiency is 92%. Shared predecessor and successor sets provide a structural similarity measure, not just a task-length-only grouping rule. The eight-point advantage over PSO+DBCA and the larger margins over the rest of the methods indicate that the groups preserve local workflow relations more effectively. In a future experiment, this explanation can be tested using the cluster purity, inter-cluster edge count, and communication-volume metrics.

Prior to ours, 270, 295, 330 cost units are consumed by the 3 algorithms. The gains are 16.7%, 23.7% and 31.8%. The number of candidate relations that are visited in search is also reduced by grouping and the repair operator guarantees that no more time is spent in infeasible particles in any later iterations. Since the provided results are in abstract cost units, a deployment--oriented validation should convert the metric to VM-hours, monetary charges, scheduler CPU time etc.

DAWSM combines DAG analysis, dependency similarity, DBTC clustering, discrete particle encoding, constraint repair, and VM allocation into a single workflow scheduler. The comparison reported is biased towards DAWSM in terms of satisfying dependencies, execution time, efficiency of clustering, and computing cost. The should be interpreted as limited to aggregated simulation results and extended through workflow-wise results, communication-cost tests, convergence plots, and repeated-run statistics.

The consistent table and plot results show dependency-satisfaction gains of 7, 11, and 17 percentage points; execution-time reductions of 16.0, 20.8, and 27.6 percent; and cost savings of 16.7, 23.7, and 31.8 percent. The tables, plots, and discussion use the same baseline values and are authoritative for these comparisons.

C SMMSF Results

TABLE V. SMMSF Results

Metric	SMMSF	SRA	SOPSO	CLUSTERACO
Makespan ms	4100	4800	5000	5600
Energy kWh	210	260	280	320
Cost units	120	150	165	200
Utilization %	94	85	82	70

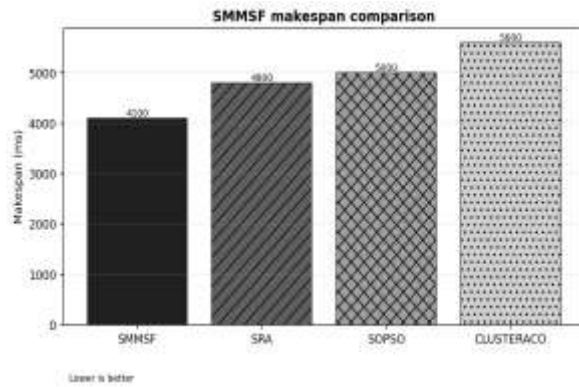


Fig. SMMSF makespan.

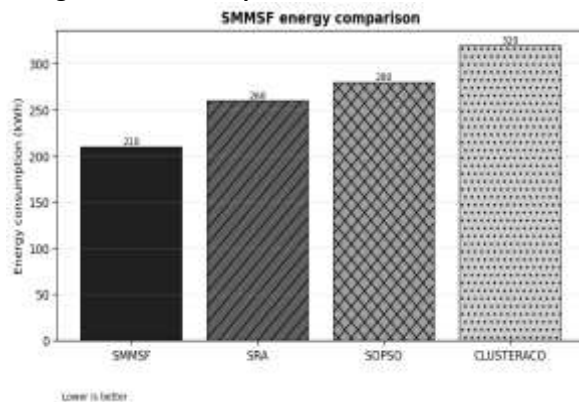


Fig. SMMSF energy.

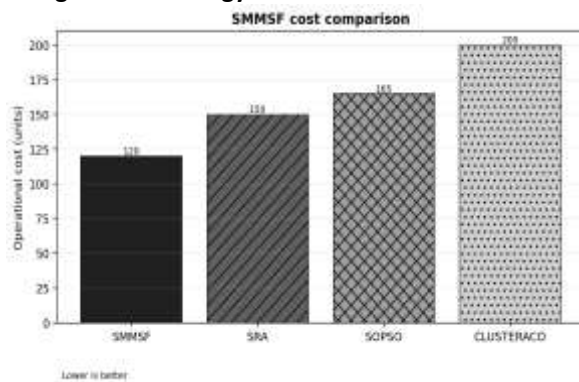


Fig. SMMSF operational cost.

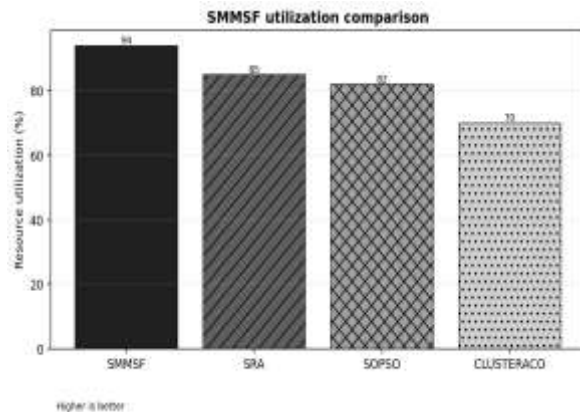


Fig. SMMSF resource utilization.

SMMSF achieves a makespan of 4100 ms. The improvements over SRA, SOPSO and CLUSTERACO are 14.6%, 18.0% and 26.8%. This benefit comes from pre-clustering compatible VMs and considering load in the objective function. The scheduler avoids both capability mismatch as well as over-concentration on a few fast machines.

Energy consumption is reduced to 210 kWh (SRA: 260 kWh, SOPSO: 280 kWh, CLUSTERACO: 320 kWh). These values represent savings of 19.2%, 25.0%, and 34.4%. The energy term discourages assignments that only shorten execution at the expense of high power machines. Since energy and makespan can be conflicting objectives, the simultaneous improvement in both (as shown in Figures 5.3 and 5.4) suggests a better mapping, not just a simple delay-energy trade-off.

The framework proposed in this paper shows 120 operational cost units. SRA, SOPSO and CLUSTERACO require 150, 165 and 200 units. The savings are 20.0%, 27.3% and 40.0% respectively. Cost needs to be normalised before weighted aggregation so that a large numerical range in one cannot dominate the fitness function. Sensitivity analysis of the cost coefficient remains a worthwhile extension, especially for spot-price variation and multi-cloud scheduling.

The resource utilization reaches 94%, which is 9, 12 and 24 percentage points higher than baselines. This shows that clustering does not just eliminate resources from the active demand usable capacity match. Monitoring refreshes the validity, allowing for overloaded or unavailable VMs to be filtered out while compatible machines that are underutilized can take on work.

SMMSF integrates dynamic monitoring, VM clustering, normalized multi-objective evaluation, adaptive search, feasibility repair, and dispatch feedback. The provided results demonstrate simultaneous improvement in makespan, energy, cost, and utilization. The main validity boundaries are rigid preference weights, aggregated simulation outputs, and lack of production-cloud measurements. Weight sensitivity, Pareto-front analysis, repeated experiments, and real workload traces are recommended to be part of next evaluation cycle.

IX DISCUSSION AND VALIDITY

The models represent different workloads and should not be compared as if they were all derived from a single general experiment. Their shared architectural outcome is that initial structure reduction combined with constraint-aware mapping enhances the metrics significant to each workload class.” QTSF provides service differentiation, DAWSM preserves workflow semantics during search, and SMMSF prevents raw units or a few fast VMs from taking over large-scale allocation.

The reported aggregate values do not isolate every component. Full ablation would require eliminating resource refinement, load repair, dependency clustering, precedence repair, VM clustering, monitoring, and adaptive exploration one by one. Level-free logs should show convergence, scheduling overhead, cluster stability, number of migrations, SLA violations, and workflow-related performance indicators.

In a production implementation, the architecture can be placed in the broker. Admission processing extracts a task profile or parses a DAG. Monitoring provides capacity, utilization, power, and price. QTSF is a model selector that selects QTSF for independent QoS jobs, DAWSM for workflows, and SMMSF for large batches with multiple

objectives. Each optimizer must operate within a time budget, keep a feasible incumbent, and, if the monitoring is out of date, revert to a deterministic heuristic.

Internal validity is an issue due to aggregate reporting. Since individual runs are not observed, seed variability and statistical significance cannot be assessed. The construct validity is contingent on the precise simulator definitions of balance, clustering efficiency, energy, cost, normalization bounds, power curves, and price functions.

The external validity is limited since the CloudSim and WorkflowSim models ignore network contention, virtualization overhead, failures, billing rules, and monitoring delay. Montage, CyberShake, Epigenomics, and Inspiral do provide valuable graph templates, but are not indicative of all production workloads. Still, tests on real infrastructure and traces are necessary.

Validity of conclusions is a function of the baseline configuration. Metaheuristics are sensitive to the population size, stopping rules and initialization. A fair comparison would specify a common evaluation budget, matched feasibility rules, a declared tuning protocol, and a agreed protocol for repeated paired seeds. The results are encouraging descriptive evidence and nothing that should replace that protocol.

X CONCLUSION

This paper unifies three cloud-scheduling models within a common control architecture. QTSF schedules independent tasks by normalizing QoS attributes, refining task clusters by resource demand, ranking ready tasks by priority density, mapping them to capability-compatible VMs, and repairing load imbalance. DAWSM extends this process to workflow DAGs by grouping related tasks, decoding particle positions in legal topological order, and repairing precedence and overload violations. SMMSF addresses large workloads by monitoring and clustering heterogeneous VMs, normalizing makespan, energy, cost, and utilization, and refining feasible mappings through adaptive population search. The models share a common observation-optimization-repair loop while preserving workload-specific requirements for differentiated service, workflow correctness, and multiobjective scalability.

The reported comparisons are in favour of the proposed approaches across all the provided aggregated metrics. QTSF achieves the lowest makespan and the highest balance and utilization among the three baselines. DAWSM achieves better dependency satisfaction and clustering efficiency, as well as less workflow time and cost. SMMSF achieves a simultaneous improvement in makespan, energy, and cost with the highest utilization. These comparisons corroborate the proposed mechanisms under the specified simulator settings and are not claimed to be that of universal superiority as run-level distributions, ablations, sensitivity tests, and production measurements are not accessible in the originating evidence.

The combined evidence supports each of the proposed models on its particular metrics, and the formal analysis proves termination and feasibility under clearly stated conditions. Future work should release executable implementations and run-level data, perform ablation and weight-sensitivity tests, report Pareto fronts and scheduling overhead, and validate the framework with public traces and production infrastructure.

REFERENCES

- [1] G. Singh and A. K. Chaturvedi, "A cost, time, energy-aware workflow scheduling using adaptive PSO algorithm in a cloud-fog environment," *Computing*, vol. 106, pp. 3279–3308, 2024, doi: 10.1007/s00607-024-01322-w.
- [2] K. Kaur, S. Garg, G. S. Aujla, N. Kumar, and A. Y. Zomaya, "A multi-objective optimization scheme for job scheduling in sustainable cloud data centers," *IEEE Transactions on Cloud Computing*, vol. 10, no. 1, pp. 172–186, 2022, doi: 10.1109/TCC.2019.2950002.
- [3] K. Dubey, S. C. Sharma, and M. Kumar, "A novel multi-objective CR-PSO task scheduling algorithm with deadline constraint in cloud computing," *Sustainable Computing: Informatics and Systems*, vol. 32, 2021.
- [4] S. A. Murad, R. M. Yusof, and S. N. M. Mohamad, "A review on job scheduling techniques in cloud computing and priority-based scheduling," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, 2022.
- [5] A. Pradhan, S. K. Bisoy, and M. R. Patra, "A survey on PSO-based meta-heuristic scheduling mechanisms in cloud computing," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 8, pp. 4888–4901, 2022.
- [6] X. Chen et al., "A WOA-based optimization approach for task scheduling in cloud computing systems," *IEEE Systems Journal*, vol. 14, no. 3, pp. 3117–3128, 2020, doi: 10.1109/JSYST.2019.2960088.

- [7] Y. Wang and X. Q. Zuo, "An effective cloud workflow scheduling approach combining PSO and idle time slot-aware rules," *IEEE/CAA Journal of Automatica Sinica*, vol. 8, no. 5, pp. 1079–1094, 2021, doi: 10.1109/JAS.2021.1003982.
- [8] S. H. Anbarkhan, H. A. Ali, and A. M. Khedr, "An enhanced PSO algorithm for scheduling workflow applications in cloud computing environments," *Electronics*, vol. 12, no. 12, 2023.
- [9] V. Sharma and M. Bala, "An improved task allocation strategy in cloud using modified K-means clustering technique," *Egyptian Informatics Journal*, vol. 21, no. 4, pp. 201–208, 2020, doi: 10.1016/j.eij.2020.02.001.
- [10] D. Yu, Y. Ying, L. Zhang, C. Liu, X. Sun, and H. Zheng, "Balanced scheduling of distributed workflow tasks based on clustering," *Knowledge-Based Systems*, vol. 199, 2020, doi: 10.1016/j.knosys.2020.105930.
- [11] M. Hosseini Shirvani and R. Noorian Talouki, "Bi-objective scheduling algorithm for scientific workflows on cloud computing platform with makespan and monetary cost minimization approach," *Complex & Intelligent Systems*, vol. 8, pp. 1085–1114, 2022, doi: 10.1007/s40747-021-00528-1.
- [12] K. K. Chakravarthi, L. Shyamala, and V. Vaidehi, "Budget-aware scheduling algorithm for workflow applications in IaaS clouds," *Cluster Computing*, vol. 23, no. 4, pp. 3405–3419, 2020, doi: 10.1007/s10586-020-03095-1.
- [13] P. Han, C. Du, J. Chen, X. Du, and M. Guizani, "Cost and makespan scheduling of workflows in clouds using list multiobjective optimization technique," *Journal of Systems Architecture*, vol. 112, 2021, doi: 10.1016/j.sysarc.2020.101837.
- [14] Z. Tong, F. Ye, B. Liu, J. Cai, and J. Mei, "DDQN-TS: A novel bi-objective intelligent scheduling algorithm in the cloud environment," *Neurocomputing*, vol. 455, pp. 419–430, 2021, doi: 10.1016/j.neucom.2021.05.070.
- [15] G. Rjoub, J. Bentahar, and O. A. Wahab, "Deep and reinforcement learning for automated task scheduling in large-scale cloud computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 6, pp. 1474–1487, 2020.
- [16] S. Mangalampalli, G. R. Karri, S. N. Mohanty, S. Ali, A. M. Alamri, and S. A. Alqahtani, "Efficient hybrid DDPG task scheduler for HPC and HTC in cloud environment," *IEEE Access*, vol. 12, pp. 108897–108920, 2024, doi: 10.1109/ACCESS.2024.3435914.
- [17] L. Zhang, Y. Wang, X. Li, and J. Liu, "Efficient prediction of makespan matrix workflow scheduling in heterogeneous cloud computing environments," *Journal of Grid Computing*, vol. 21, 2023, doi: 10.1007/s10723-023-09711-9.
- [18] A. Tarafdar and V. K. Prasanna, "Energy and makespan aware scheduling of deadline-sensitive tasks in the cloud environment," *Journal of Grid Computing*, vol. 19, 2021, doi: 10.1007/s10723-021-09548-0.
- [19] Z. Sun, B. Zhang, C. Gu, R. Xie, B. Qian, and H. Huang, "ET2FA: A hybrid heuristic algorithm for deadline-constrained workflow scheduling in cloud," *IEEE Transactions on Services Computing*, vol. 16, no. 3, pp. 1807–1821, 2023, doi: 10.1109/TSC.2022.3196620.
- [20] S. A. Alsaady, A. D. Abbood, and M. A. Sahib, "Heuristic initialization of PSO task scheduling algorithm in cloud computing," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 6, pp. 2370–2382, 2022, doi: 10.1016/j.jksuci.2020.11.002.
- [21] S. Qin, D. Pi, Z. Shao, and Y. Xu, "Hybrid collaborative multi-objective fruit fly optimization algorithm for scheduling workflow in cloud environment," *Swarm and Evolutionary Computation*, vol. 68, 2022, doi: 10.1016/j.swevo.2021.101008.
- [22] A. Pradhan, S. K. Bisoy, and M. R. Patra, "Intelligent decision-making of load balancing using deep reinforcement learning and parallel PSO in cloud environment," *IEEE Access*, vol. 10, pp. 77413–77432, 2022, doi: 10.1109/ACCESS.2022.3192628.
- [23] S. Mangalampalli, G. R. Karri, S. K. Swain, and G. S. Rao, "Multi-objective prioritized workflow scheduling using deep reinforcement-based learning in cloud computing," *IEEE Access*, vol. 12, pp. 5621–5638, 2024, doi: 10.1109/ACCESS.2024.3350741.
- [24] S. Mangalampalli, G. R. Karri, S. K. Swain, and G. S. Rao, "Multi-objective trust-aware task scheduling algorithm in cloud computing using whale optimization," *Journal of King Saud University - Computer and Information Sciences*, vol. 35, no. 2, pp. 609–624, 2023.
- [25] H. Liang, B. Ding, Y. Du, and F. Li, "Parallel optimization of QoS-aware big service processes with discovery of skyline services," *Future Generation Computer Systems*, vol. 125, pp. 496–514, 2021, doi: 10.1016/j.future.2021.06.020.
- [26] K. A. Tabary, H. Motameni, B. Barzegar, and E. Akbari, "QoS aware task scheduling using hybrid genetic algorithm in cloud computing," *IEEE Access*, vol. 12, 2024, doi: 10.1109/ACCESS.2024.3520412.

- [27] M. A. Rakrouki, T. Alharbi, and T. Alghamdi, "QoS-aware algorithm based on task flow scheduling in cloud computing," *Sensors*, vol. 22, no. 7, 2022, doi: 10.3390/s22072632.
- [28] Z. Wang, Y. Zhao, M. Liu, R. Chen, and X. Zhou, "Reinforcement learning based task scheduling for cloud-edge collaboration," *Journal of Cloud Computing*, vol. 12, 2023, doi: 10.1186/s13677-023-00553-0.
- [29] K. K. Chakravarthi, L. Shyamala, and V. Vaidehi, "Reliable budget-aware workflow scheduling strategy on multi-cloud environment," *Cluster Computing*, vol. 25, no. 2, pp. 1189–1205, 2022, doi: 10.1007/s10586-021-03464-4.
- [30] A. K. Maurya, "Resource and task clustering based scheduling algorithm for workflow applications in cloud computing environment," presented at the 2020 6th International Conference on Parallel, Distributed and Grid Computing (PDGC), 2020, pp. 86–91. doi: 10.1109/PDGC50313.2020.9315806.
- [31] A. Song, W. N. Chen, X. Luo, Z. H. Zhan, and J. Zhang, "Scheduling workflows with composite tasks: A nested particle swarm optimization approach," *IEEE Transactions on Services Computing*, vol. 15, no. 2, pp. 1074–1088, 2022, doi: 10.1109/TSC.2020.2975774.
- [32] E. H. Houssein, A. G. Gad, Y. M. Wazery, and P. N. Suganthan, "Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends," *Swarm and Evolutionary Computation*, vol. 62, 2021, doi: 10.1016/j.swevo.2021.100841.
- [33] M. Menaka and K. S. Sendhil Kumar, "Workflow scheduling in cloud environment-Challenges, tools, limitations and methodologies: A review," *Measurement: Sensors*, vol. 24, 2022, doi: 10.1016/j.measen.2022.100436.
- [34] S. E. Shukri, R. Al-Sayyed, A. Hudaib, and S. Mirjalili, "Enhanced multi-verse optimizer for task scheduling in cloud computing environments," *Expert Systems with Applications*, vol. 168, Art. no. 114230, 2021, doi: 10.1016/j.eswa.2020.114230.
- [35] P. Pirozmand, A. M. Rahmani, A. Rezaei, and E. Farrokhi, "Multi-objective hybrid genetic algorithm for task scheduling problem in cloud computing," *Neural Computing and Applications*, vol. 33, pp. 13075–13088, 2021, doi: 10.1007/s00521-021-06002-w.
- [36] M. S. Ajmal, Z. Iqbal, F. Z. Khan, M. Ahmad, I. Ahmed, and B. B. Gupta, "Hybrid ant genetic algorithm for efficient task scheduling in cloud datacenters," *Computers & Electrical Engineering*, vol. 95, Art. no. 107419, 2021, doi: 10.1016/j.compeleceng.2021.107419.
- [37] H. Emami, "Cloud task scheduling using enhanced sunflower optimization algorithm," *ICT Express*, vol. 8, no. 1, pp. 97–100, 2022, doi: 10.1016/j.icte.2021.08.001.
- [38] L. Abualigah and A. Diabat, "A novel hybrid antlion optimization algorithm for multi-objective task scheduling problems in cloud computing environments," *Cluster Computing*, vol. 24, no. 1, pp. 205–223, 2021, doi: 10.1007/s10586-020-03140-2.