Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.084063

ARTICLE

Hybrid Fuzzy Spark Lion Whale Algorithm for Energy-Efficient Resource Allocation and Task Migration in Cloud Data Centers

Nidhika Chauhan[1,*], Navneet Kaur[1], Jawad Khan[2], Younhyun Jung[2], Haleem Farman[3], Ahmed Sedik[3,4] and Sohaib Bin Altaf Khattak[3]

[1]Department of Computer Science and Engineering, Chandigarh University, Gharuan, Mohali, Punjab, India
[2]School of Computing, Gachon University, Seongnam, Republic of Korea
[3]Smart Systems Engineering Laboratory, College of Engineering, Prince Sultan University, Riyadh, Saudi Arabia
[4]Department of Robotics and Intelligent Machines, Faculty of Artificial Intelligence, Kafrelsheikh University, Kafrelsheikh, Egypt

*Corresponding Author: Nidhika Chauhan. Email: nidhi29.chauhan@gmail.com

Received: 17 April 2026; Accepted: 24 July 2026; Published: 15 September 2026

ABSTRACT: The exponential growth of cloud data centers necessitates highly efficient resource allocation and task migration strategies. However, multi-dimensional memory fragmentation severely limits the efficacy of standard scheduling algorithms under heavy-tailed, real-world workloads. This paper proposes Fuzzy-SLW, a hybrid swarm-intelligence architecture that integrates a Mamdani fuzzy-inference pre-filter with a distributed Spark Lion-Whale Optimization (SLWO) core via Apache Spark. The fuzzy pre-filter mathematically prunes the search space using non-compressible hardware constraints, while the Spark execution model resolves the traditional serial bottleneck of swarm intelligence. Evaluated within a discrete-event environment utilizing the Google Cluster Trace (2019), Fuzzy-SLW demonstrates a greater than 240% relative improvement (+42.2 percentage points) in virtual machine utilization over load-scattering metaheuristics and avoids the premature policy convergence observed in Deep-DQN baselines. For large-population offline optimization configurations ($P \geq 5000$ individuals), the distributed architecture achieves a 5.85 times sub-linear Amdahl speedup; below this population threshold, including the $P = 20$ configuration used for online, per-task scheduling, thread-pool context-switching overhead dominates and distributed partitioning does not improve wall-clock latency. The results empirically quantify the necessary tradeoff between aggressive hardware consolidation and Service Level Agreement preservation, establishing Fuzzy-SLW as a scalable solution for power-constrained hyper-scale environments.

KEYWORDS: Cloud computing; resource allocation; task migration; fuzzy logic; metaheuristic optimization; energy efficiency; virtual machine placement; deep reinforcement learning; Apache Spark

1 Introduction

Cloud computing has revolutionized the way computational services are delivered, enabling on-demand, elastically scalable access to infrastructure, platform, and software resources across shared physical substrates. The concept originated in the telecommunications industry during the 1990s, where Virtual Private Network services demonstrated the feasibility of dynamically redirecting capacity across shared circuits, and has since grown to underpin virtually every category of enterprise and scientific workload. This growth carries a significant consequence: cloud data centers collectively account for a substantial and rising fraction of global electricity consumption, elevating energy efficiency to a primary engineering objective alongside performance and reliability [1].

Resource allocation is a critical issue in the efficient functioning of cloud data centres, defined as the process of assigning the workload of different users to physical and virtual resources in a systematic manner to maximise overall resource utilisation, minimise energy consumption, and maintain service-level agreements [2]. In the Infrastructure-as-a-Service (IaaS) model, this corresponds to allocating CPU time, memory, storage, and network bandwidth to virtual machines (VMs), the main abstraction on which users depend to consume computational resources. This is further complicated by the ever-moving, non-stationary nature of cloud workloads, in which the demand changes over time and spans from milliseconds to days, as well as heavy-tailed, heterogeneous task distributions, which make any set allocation policy structurally inadequate [1].

Virtual machine migration plays a crucial role in enabling data center operators to respond to workload dynamics. This process involves the live transfer of memory state, CPU context, and disk contents of a VM from a source physical machine (PM) to a destination PM, allowing operators to rebalance load, consolidate underutilized servers into low-power states, and maintain service continuity in the face of hardware faults [3]. Task migration extends this concept to the scheduling layer, enabling the transparent relocation of pending or executing jobs to alternative VM instances in the interest of fault tolerance, load balance, and performance optimization [4]. Despite their practical importance, accurately modeling the combined costs of these mechanisms in simulation remains challenging because live migration incurs measurable network transfer latency and energy overhead that conventional simulators often approximate or neglect.

In a heterogeneous cloud setting, the vast search space size results in premature convergence with the conventional metaheuristic optimization methods such as the Differential Evolution (DE) and Particle Swarm Optimization (PSO). The previous fuzzy controllers could not adapt to the changes in workloads for different machine configurations, and used a fixed rule base [5]. High-fidelity multi-objective optimization is often not tractable when dealing with VM placement in production environments due to real-time latency requirements [6].

While Deep Reinforcement Learning (DRL) agents offer adaptive scheduling, they frequently experience premature policy convergence when navigating the heavy-tailed heterogeneity of real-world cloud traces. Addressing these challenges simultaneously requires a framework that combines principled search-space reduction, distributed parallel execution, and physically grounded simulation.

To address these limitations, this paper proposes the Fuzzy Spark Lion Whale (Fuzzy-SLW) algorithm, a novel hybrid optimization framework for resource allocation and task migration in cloud data centers. As illustrated in Fig. 1, the target scenario encompasses the full lifecycle of task execution from initial scheduling through fault-triggered migration. Fuzzy-SLW provides a training-free, deterministic search-space reduction via a Mamdani fuzzy-inference pre-filter, which routes tasks exclusively to resource-tier-compatible PMs using non-compressible hardware constraints. The Spark Lion Whale Optimization (SLWO) core then identifies the optimal placement within that filtered subspace via parallel Apache Spark partitions, enabling sub-linear scaling of computational cost with workload volume. This distributed parallelism is beneficial only above a critical population threshold: empirically, partitioning overhead is amortized and sub-linear Amdahl speedup is realized at $P \geq 5000$ individuals, whereas for the smaller populations characteristic of online, per-task scheduling (e.g., $P = 20$), thread-pool context-switching overhead dominates and parallel execution does not reduce wall-clock latency. Accordingly, the Spark execution model is recommended specifically for large-population, offline optimization configurations, with its behavior at smaller populations treated as a system boundary condition rather than a general-purpose scheduling mechanism.

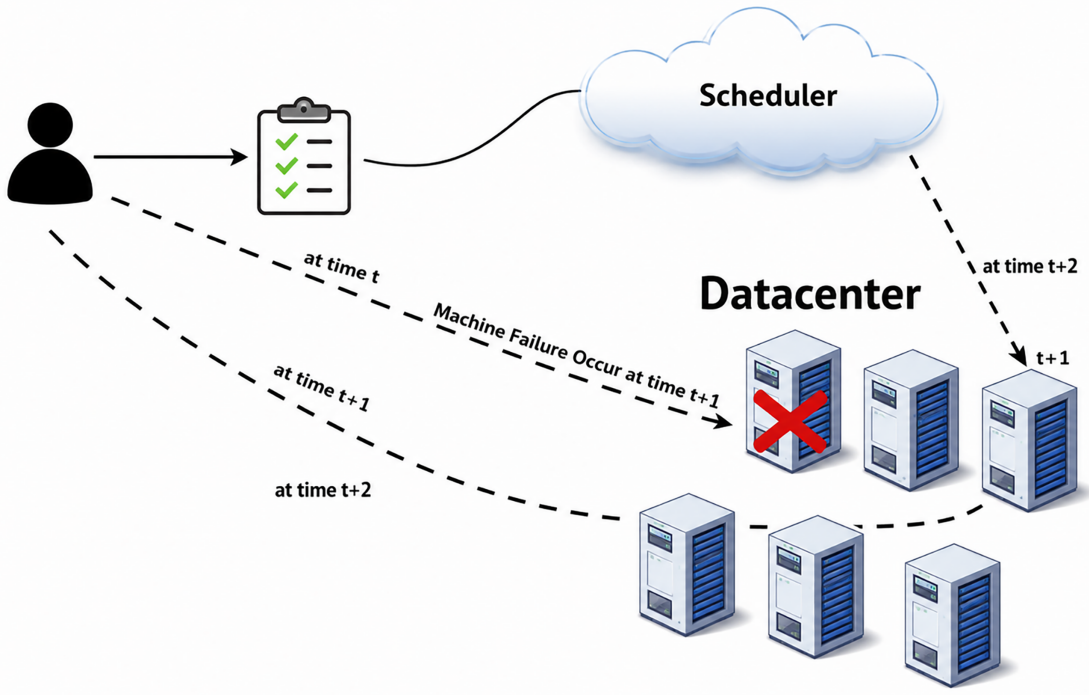

Figure 1: Server architecture for task migration in cloud computing environments. When a user submits a task at time t, the cloud scheduler allocates a physical machine. If the scheduler detects failure of a machine at time $t + 1$, it initiates the task migration process, with the result that the machine failure is not visible to the application, allowing the service to continue while the task is moved to a different machine.

Our Contributions

The principal contributions of this work are as follows.

- A Mamdani fuzzy inference pre-filter that exploits the space-shared non-compressibility of RAM and disk to prune the 800-PM search space to approximately 267 tier-matched candidates per task, concentrating the metaheuristic search on a high-relevance subspace.
- A nine-dimensional composite fitness formulation for SLWO that captures delay, power, resource utilization, storage, energy efficiency, bandwidth, CPU load, memory headroom, and availability as a single multi-objective ranking signal.
- A distributed Spark execution model that resolves the serial swarm bottleneck through RDD-based population partitioning, confirmed by an empirical Amdahl benchmark showing 5.85 times sub-linear speedup at $K = 8$ partitions for large-population offline optimization configurations ($P \geq 5000$); below this population threshold, thread-pool context-switching overhead dominates and parallelism does not reduce wall-clock latency, a system boundary condition discussed in Section 5.5.

To validate the proposed architecture, a dual-dataset evaluation was conducted utilizing a Standardized Uniform Benchmark and the Google Cluster Trace (2019) [7]. The framework is benchmarked against load-scattering metaheuristics, modern swarm optimizers, and state-of-the-art Deep Reinforcement Learning schedulers including Deep-DQN and Deep-A2C. The evaluation explicitly quantifies the fundamental tradeoff between aggressive density-maximization and SLA preservation, establishing the system boundaries of the proposed architecture.

The remainder of this paper is organized as follows. Section 2 surveys related work. Section 3 presents the proposed Fuzzy-SLW architecture. Section 4 describes the experimental setup and workload modeling. Section 5 reports performance results. Section 6 discusses limitations and system boundaries, and Section 7 concludes the paper.

2 Related Work

Cloud infrastructure, including auto-scaling, elastic provisioning of resources, and server consolidation, has been the focus of ongoing research for the last ten years [8]. The discussion is organized under three thematic axes that together define the landscape for Fuzzy-SLW.

2.1 Metaheuristic and Swarm Intelligence Scheduling

Beloglazov and Buyya [9] proposed dynamic heuristic algorithms which consider real-time CPU utilization signals to mitigate migration overhead and increase server efficiency. In contrast, Talwani and Singla [10] addressed the energy overhead of VM migration by proposing an enhanced Artificial Bee Colony approach that minimizes overall energy consumption and migration count; however, the greedy decision structure of such metaheuristic approaches prevents globally optimal allocation under varying loads and can restrict applicability in latency-sensitive systems.

Zhang et al. [11] formulated multi-VM migration scheduling as a combinatorial optimization task and demonstrated improvements over greedy baselines in migration time and resource consumption. Arya et al. [12] suggested a dynamic resource management policy that changes the migration behavior based on perceived workload change, but the change is reactive and not anticipatory.

Sutar et al. [13] have used Ant Colony Optimization for live VM migration and compared its performance with threshold-based policies in terms of resource efficiency and energy savings, showing that the ACO-based approach is superior. Wang et al. [14] proposed a decentralized multi-agent architecture based on auction and negotiation to reduce migration overhead in Internet-of-Things systems.

For VM-dense deployments, Nagpure et al. [15] proposed a dynamic allocation scheme which minimises resource wastage using skewness. Xu et al. [16] proposed a method called Energy-conscious Resource Allocation for scientific workflow scheduling, which had significant energy savings over traditional scientific workflow engines.

In the field of hybrid metaheuristic literature, Chauhan and Agrawal [17] designed Optimized Kernel Naive Bayesian system for cloud resource selection that provides accurate resource allocation using probabilistic resource model in hybrid metaheuristics. In the context of edge-cloud continua, as well as IIoT deployments, Hosny et al. [18] proposed a refined whale optimization algorithm for multi-user dependent task offloading. In VM resource allocation, Shi and Lin [19] used multi-objective genetic algorithms (MOGA) to stabilize the multi-VM distributions over long operation periods. To improve the performance of the conventional schemes in terms of resource utilization metrics, a hybrid method called the RAFL framework based on Phasor Particle Swarm Optimization and Dragonfly Algorithm was proposed by Thakur and Goraya [20].

Hybrid methods combining the Whale Optimization Algorithm with Moth-Flame Optimization leverage deterministic spiral-path exploitation to counterbalance stochastic bubble-net exploration, and are included as a baseline in the present evaluation to position Fuzzy-SLW within this lineage.

More recent hybrid metaheuristics continue this trajectory: Gopu et al. [21] applied the NSGA-III multi-objective evolutionary algorithm to distributed-cloud VM placement to jointly reduce resource wastage, power consumption, and network delay; Keshri and Vidyarthi [22] combined Ant Colony Optimization

with Grey Wolf Optimization for communication-aware VM placement, reporting improvements in power consumption, resource wastage, and bandwidth utilization over single-metaheuristic and greedy baselines; and Mehrabadi et al. [23] proposed an improved Harris Hawks Optimization algorithm for energy-aware VM placement in cloud data centers. These works confirm that hybrid swarm-intelligence pairings remain an active and productive direction for energy-efficient cloud resource allocation.

Despite these advances, purely swarm-based approaches commonly treat fitness evaluation as a serial operation, creating a computational bottleneck that limits scalability to hyper-scale task volumes.

2.2 Deep Reinforcement Learning in Resource Allocation

Tuli et al. [6] introduced HUNTER, a multi-objective AI-based data center energy management system that achieves state-of-the-art energy and SLA performance by integrating predictive modeling with online scheduling. Deep Q-Networks with experience replay and target networks have also been used to achieve competitive scheduling performance in dynamic cloud environments.

Under heavy-tailed task distributions, an Advantage Actor-Critic scheduler with entropy regularization can prevent the premature convergence that is a common issue with value-based methods.

The entropy regularization in A2C prevents the policy from collapsing onto a small set of high-reward actions, maintaining exploration even when the workload distribution shifts. These results motivate the inclusion of both DRL paradigms as baselines in the present evaluation. It is noted that DRL schedulers require substantial offline training on historical trace data before deployment, whereas Fuzzy-SLW operates deterministically without any training phase, which is a relevant distinction for systems that must respond to previously unseen workload distributions.

2.3 Limitations of Current Evaluation Methodologies

SDN and NFV have enabled a new set of resource allocation techniques that leverage the centralized, real-time awareness of network state across the entire network to guide placement decisions [24,25].

The SDN control plane decouples network intelligence from the network forwarding hardware, and the NFV abstraction removes the need for dedicated network appliances by instead allowing the functions to be instantiated, migrated and scaled on the commodity infrastructure, providing an orchestrator with visibility into link utilization, queue occupancy, and the energy state of devices that does not exist in a compute-layer scheduler.

Building on this capability, Montazerolghaem et al. [26] proposed GreenVoIP, an NFV/SDN-based energy-efficient resource allocation framework for virtualized cloud multimedia centers that dynamically activates and deactivates VoIP servers and network equipment in response to load, demonstrating that joint control of the network and compute layers reduces the number of active devices and the associated energy draw without violating service quality constraints. This line of work establishes that energy-efficient resource allocation is not confined to the compute layer of the data center: substantial savings are also available at the network layer when the SDN controller exposes sufficiently fine-grained telemetry to the allocation algorithm.

A related body of work applies hybrid swarm-intelligence optimizers to SDN-coordinated task scheduling in fog-assisted Internet-of-Things networks. Salehnia et al. [27] proposed an SDN-based optimal task scheduling method for Fog-IoT networks that combines the Aquila Optimizer with the Whale Optimization Algorithm (AO-WOA) to allocate fog-computing resources to IoT task requests, using a centralized SDN controller layer to coordinate network elements and reduce task completion time, makespan, and traffic overhead relative to single-optimizer and threshold-based baselines. The AO-WOA design is structurally

relevant to the present work: it pairs a global-exploration metaheuristic with a local-exploitation metaheuristic under the coordination of a centralized SDN controller, mirroring the division of labor between the whale (exploration) and lion (exploitation) operators in the SLWO core proposed here, and confirming that SDN-coordinated hybrid swarm scheduling is an established and effective pattern in resource-constrained, latency-sensitive network-edge environments.

Fuzzy-SLW is architecturally compatible with both of these directions: the fuzzy inference engine ingests normalized RAM and disk telemetry of the type exported by SDN management planes, and the SLWO fitness function incorporates SDN-reported flow-level bandwidth utilization as the B component of the nine-dimensional objective. Unlike GreenVoIP, which targets network-equipment activation, and AO-WOA, which targets fog-layer task placement, Fuzzy-SLW is positioned at the data-center compute layer, suggesting that the three approaches are complementary rather than competing and could, in principle, be composed within a single SDN-orchestrated control plane spanning network equipment, fog nodes, and cloud physical machines.

A structural limitation cuts across nearly all surveyed works: current evaluations predominantly utilize time-shared resource abstractions, which allow CPU and memory to be oversubscribed without triggering allocation failures. This can obscure the execution-delay penalties inherent in aggressive consolidation, because oversubscription masks the Out-of-Memory rejection events that space-shared environments expose. This study addresses this gap by utilizing rigid space-shared provisioning and exposes the operational SLA penalties inherent to density-maximizing policies.

3 Proposed Methodology and System Architecture

This section presents the complete Fuzzy-SLW architecture, beginning with the system model and problem formulation, proceeding through the Mamdani fuzzy pre-filter and multi-objective fitness function, and concluding with the distributed Spark execution model and computational complexity analysis.

3.1 System Model and Problem Formulation

The simulated data center comprises $N = 800$ heterogeneous physical machines $\mathcal{M} = \{m_1, \ldots, m_N\}$. Each PM m_i is characterized by its RAM capacity $x_i \in \{1024, 2048, 4096\}$ GB, disk storage $y_i \in \{10{,}000, 15{,}000, 20{,}000\}$ GB, CPU capacity measured in MIPS, and a power model derived from empirical cloud benchmarks [9]. Virtual machines are generated as discrete tasks T with resource demand profiles $\langle r_{\text{RAM}}, r_{\text{Disk}}, r_{\text{MIPS}} \rangle$ drawn from the configured workload trace. The allocation problem is to assign each incoming task T to a physical machine $m^* \in \mathcal{M}$ such that the nine-dimensional fitness objective $F_{\text{SLWO}}(m^*)$ is maximized subject to the capacity constraints $r_{\text{RAM}} \leq x_{m^*}$, $r_{\text{Disk}} \leq y_{m^*}$, and $r_{\text{MIPS}} \leq \text{MIPS}_{m^*}$. This is a variant of the vector bin-packing problem, which is NP-hard in the general case [28]. The task migration sub-problem is triggered when the CPU utilization of the hosting PM exceeds a static threshold, requiring a re-allocation of the task to an alternative host while minimizing migration latency and energy overhead.

3.2 Mamdani Fuzzy-Inference Pre-Filter

The fuzzy pre-filter operates as a Mamdani inference engine [29] over two input dimensions: RAM and disk storage. The selection of these two dimensions, rather than CPU or bandwidth, is grounded in the fundamental distinction between space-shared and time-shared resources in cloud hypervisors. CPU cycles and network bandwidth are elastic: they can be throttled, time-multiplexed, or oversubscribed under hypervisor policies without triggering immediate task failure. Memory and disk storage are space-shared non-compressible resources; if a physical machine lacks sufficient RAM to host a VM, the hypervisor triggers an Out-of-Memory event that terminates the task without recovery. The pre-filter therefore eliminates

candidates that would cause hard allocation failures, while CPU and bandwidth optimization is deferred to the SLWO fitness function where their elastic behavior can be modeled continuously.

The triangular membership function is defined as

$$\mu_A(x) = \max\left(0,\ 1 - \frac{|x - c|}{w}\right), \tag{1}$$

where c is the central value and w is the half-width of the triangle. Triangular Membership Functions (MFs) were selected to minimize computational complexity. In contrast to Gaussian MFs, which involve floating-point exponential operations, Triangular MFs can be evaluated using linear arithmetic, which is important for low-latency execution requirements of cloud schedulers [29].

Compared to trapezoidal functions, the triangular form requires one fewer parameter per linguistic term, simplifying the sensitivity analysis presented in Section 5.6.

The Resource Allocation Decision (RAD) is computed via centroid defuzzification with output singletons at tier centres:

$$\text{RAD} = \frac{w_1 \cdot \mu_L + w_2 \cdot \mu_M + w_3 \cdot \mu_H}{\mu_L + \mu_M + \mu_H}, \tag{2}$$

where μ_L, μ_M, and μ_H denote membership degrees in the Low, Medium, and High categories, respectively, and the output singleton weights $w_1 = 0.25$, $w_2 = 0.50$, $w_3 = 0.75$ correspond to the tier centres in the normalized $[0,1]$ output domain. This implements standard centroid defuzzification for symmetric triangular output singletons [29].

The membership function centres are aligned to the discrete hardware tiers of the simulated environment. RAM centres at $\{0.25, 0.50, 1.00\}$ on the normalized PM scale correspond exactly to the 1024, 2048, and 4096 GB tiers, which mirror modern high-memory server configurations utilized for memory-bound enterprise workloads. The half-width $w = 0.25$ is the standard value for three equidistant triangular partitions [29], ensuring smooth tier transitions at boundaries. The average membership value n used for rule evaluation is computed as

$$n = \frac{\text{RAD}_{\text{RAM}} + \text{RAD}_{\text{Disk}}}{2}, \tag{3}$$

and machines are assigned to candidate lists according to the four-rule base in Table 1.

Table 1: Fuzzy rule base for resource allocation and task migration in Fuzzy-SLW. The average membership value $n \in [0,1]$ is computed from the fuzzified RAM and disk-storage scores of each candidate machine via Eq. (3).

Rule	Condition	Candidate Machine List
Rule 1	$0.00 \le n < 0.25$	R1D1 or R1D3
Rule 2	$0.25 \le n < 0.50$	R1D2 or R2D1
Rule 3	$0.50 \le n < 0.75$	R2D2 or R2D3
Rule 4	$0.75 \le n \le 1.00$	R3D1, R3D2, or R3D3

Fig. 2 illustrates the complete four-stage FRBS pipeline as implemented in Fuzzy-SLW.

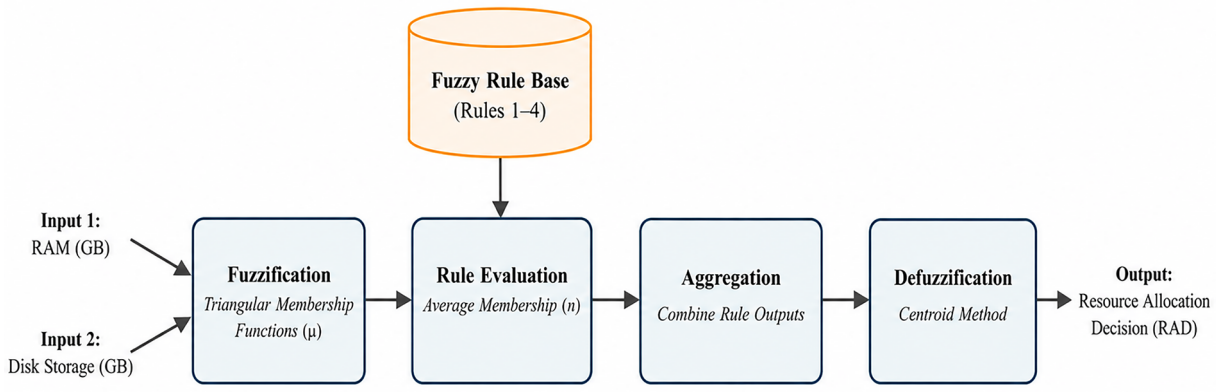

Figure 2: Schematic of the Mamdani fuzzy-inference system employed in Fuzzy-SLW. The four stages, fuzzification, rule evaluation, aggregation, and defuzzification, map crisp RAM and disk-storage inputs to a crisp Resource Allocation Decision (RAD) value that determines the candidate machine tier.

Fig. 3 illustrates the membership functions for both resource dimensions with fully labeled axes. The overlap width of 0.25 ensures that a machine near a tier boundary contributes non-zero membership to both adjacent tiers, preventing hard discontinuities in candidate selection.

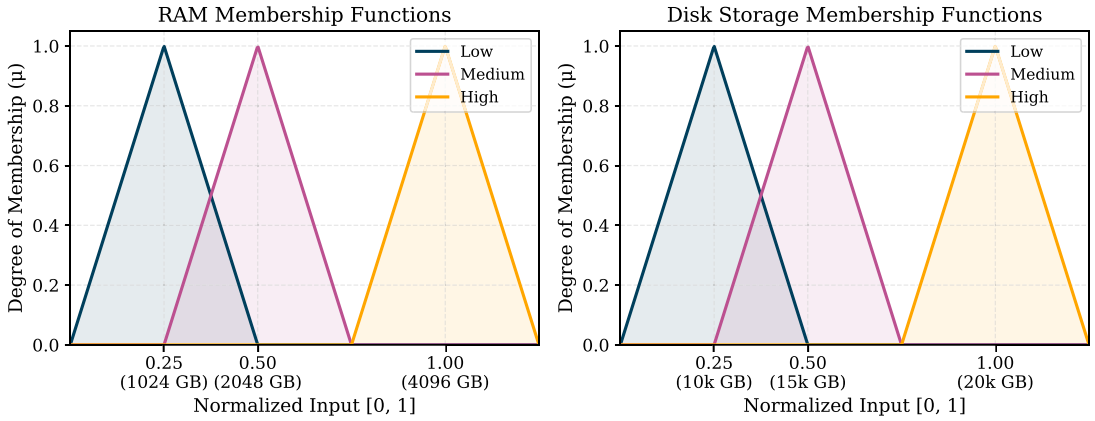

Figure 3: Triangular fuzzy membership functions for the RAM dimension (left) and disk-storage dimension (right) employed in the Fuzzy-SLW pre-filter. Horizontal axes show normalized input values in $[0, 1]$; physical hardware values in gigabytes are annotated at each tier centre. The overlap regions allow machines near category boundaries to contribute proportionally to adjacent candidate lists, producing smooth tier transitions.

The rule structure in Table 1 enforces a monotone mapping from resource demand to hardware tier, so that tasks with large resource footprints are consistently routed toward high-capacity hosts. This reduces the effective candidate pool from all 800 PMs to approximately 267 tier-matched machines, which is the principal mechanism through which the fuzzy layer reduces the search burden on the SLWO core.

3.3 Multi-Objective Defragmentation Fitness

The SLWO core evaluates each candidate machine against a nine-dimensional composite fitness function:

$$F_{\mathrm{SLWO}}(\mathbf{x}) = \sum_{k=1}^{9} w_k \cdot p_k(\mathbf{x}), \tag{4}$$

where $p_k(\mathbf{x})$ enumerates delay (D), power consumption (P), resource utilization (R), storage usage (S), energy efficiency (E), bandwidth utilization (B), CPU load (C), memory headroom (H), and availability (A) of candidate placement $\mathbf{x}$. The weight vector $(w_D, w_P, w_R, w_S, w_E, w_B, w_C, w_H, w_A) = (0.12, 0.13, 0.13, 0.10, 0.13, 0.10, 0.11, 0.10, 0.08)$ sums to unity and is calibrated against energy-weighted consolidation objectives. The three-weight LWO formulation from which SLWO is derived is

$$F(\mathbf{x}) = \alpha \cdot C(\mathbf{x}) + \beta \cdot U(\mathbf{x}) + \gamma \cdot E(\mathbf{x}), \tag{5}$$

with $\alpha = 0.4$, $\beta = 0.3$, $\gamma = 0.3$. The nine-dimensional extension in Eq. (4) captures the additional cloud-specific objectives that are absent from the base LWO formulation.

The selection of RAM and disk for the pre-filter stage, rather than including them directly in the fitness function, reflects the physical reality that these resources cannot be partially allocated: a task either fits in available RAM or it does not. Including RAM and disk as continuous fitness components would allow the optimizer to select a PM that technically scores well on Eq. (4) while still lacking sufficient physical RAM to host the task. The pre-filter eliminates this logical inconsistency before any fitness evaluation takes place.

3.4 Distributed Spark Execution Model

Apache Spark is an open-source in-memory distributed computing system that supports iterative algorithms through Resilient Distributed Dataset (RDD) abstractions, retaining intermediate results in memory between computation steps and providing automatic lineage-based fault recovery. In the SLWO implementation, the solution population is partitioned into K independent RDD slices distributed across Spark worker nodes. Each partition independently executes the lion and whale search operators on its local sub-population:

- Whale step (global exploration): each particle updates its position via the spiral bubble-net trajectory toward the current global best, implementing the Whale Optimization Algorithm's exploration phase.
- Lion step (local exploitation): each particle refines its position via a territory-search in the neighborhood of the current global best, implementing the Lion Algorithm's exploitation phase.

A global aggregation step identifies the population-wide best solution and broadcasts it to all partitions before the next iteration. This architecture eliminates the serial fitness evaluation bottleneck that constrains single-node metaheuristics and has been validated in prior Spark-based metaheuristic work [30]. Algorithm 1 presents the complete Fuzzy-SLW procedure. The deterministic convergence criterion is the maximum iteration bound $T_{\max} = 30$, which serves as a hard scheduling deadline compatible with real-time placement latency requirements [6]. The static overload threshold $\theta = 0.80$ follows the Beloglazov and Buyya policy [9], applied uniformly across all compared algorithms to ensure comparative fairness. The migration latency of 0.8 s/GB is derived from a standard 10 Gbps Top-of-Rack network topology: accounting for TCP/IP encapsulation overhead and memory page dirty-rate during the pre-copy phase, the effective transfer rate is approximately 1.25 GB/s, consistent with empirical live-migration benchmarks [31].

The complete procedure in Algorithm 1 integrates the fuzzy pre-filter, the parallel SLWO search, and the migration handler into a single pipeline whose stages map directly onto the FRBS architecture shown in Fig. 2.

3.5 Computational Complexity Analysis

The time complexity of Fuzzy-SLW decomposes into two stages. The fuzzy pre-filter evaluates N PMs per task using $O(1)$ arithmetic per machine, giving a filter cost of $O(N)$. The SLWO stage iterates over $T_{\max}$ generations with a population of P candidates drawn from the filtered set of size $|C^*| \approx N/3$. Distributing P individuals across K partitions, the total SLWO cost is $O(P \cdot T_{\max} \cdot N/(3K))$. The combined per-task complexity of Fuzzy-SLW is therefore $O(N + P \cdot T_{\max} \cdot N/(3K))$. Table 2 summarizes the complexity of all evaluated algorithms.

Table 2: Time and space complexity of all evaluated algorithms. P denotes population size, T maximum iterations, N number of physical machines, K Spark partitions, $|C|$ fuzzy-filtered candidate set ($|C| \approx N/3$), d problem dimension, $|S|$ state space size.

Algorithm	Time Complexity	Space
Fuzzy-SLW	$O(N + P \cdot T \cdot \|C\|/K)$	$O(N + P)$
SLWO	$O(P \cdot T \cdot N/K)$	$O(N + P)$
FPA	$O(P \cdot T \cdot N)$	$O(N + P)$
KB-FPA	$O(P \cdot T \cdot N)$	$O(N + P + \|\text{KB}\|)$
Firefly	$O(P^2 \cdot T \cdot N)$	$O(N + P)$
WOA-MFO	$O(P \cdot T \cdot N)$	$O(N + P)$
Deep-DQN	$O(T \cdot \|S\|)$	$O(\|S\| \cdot \|A\|)$
Deep-A2C	$O(T \cdot \|S\|)$	$O(\|S\| \cdot \|A\|)$
CMA-ES	$O(d^2 \cdot T \cdot N)$	$O(d^2)$

Algorithm 1: Fuzzy Spark Lion Whale (Fuzzy-SLW) for resource allocation and task migration

Require: PM pool $\mathcal{M}$; task T with demand $\langle r_{\text{RAM}}, r_{\text{Disk}}, r_{\text{MIPS}} \rangle$; $T_{\max}$; K partitions
Ensure: Selected machine m^*; updated allocation state
Initialize resource pool $\mathcal{M}$ and VM inventory $\mathcal{V}$
for each $m_i \in \mathcal{M}$ **do**
    Compute $\mu_{\text{RAM}}(x_i)$, $\mu_{\text{Disk}}(y_i)$ via Eq. (1)
    Compute $n_i \leftarrow (\text{RAD}_{\text{RAM}}(x_i) + \text{RAD}_{\text{Disk}}(y_i))/2$
end for
Apply Rules 1–4 (Table 1) to select candidate list C^*
Distribute C^* across K Spark partitions as initial population
$t \leftarrow 0$; $m_{\text{best}} \leftarrow \arg\max_{m \in C^*} F_{\text{SLWO}}(m)$
repeat
    for each partition $k = 1, \ldots, K$ **in parallel do**
        Whale step: update positions toward m_{best}
        Lion step: refine positions in neighborhood of m_{best}
        Evaluate $F_{\text{SLWO}}(m_i)$ for all m_i in partition (Eq. (4))
    end for

(Continued)

Algorithm 1 (continued)

15: Aggregate results; broadcast updated m_{best}
16: $t \leftarrow t + 1$
17: **until** $t = T_{\max}$
18: $m^* \leftarrow m_{\text{best}}$; execute task T on m^*
19: **if** CPU utilization of m^* exceeds $\theta = 0.80$ [9] **then**
20: Compute $\Delta t_{\text{mig}} = r_{\text{RAM}} \times 0.8$ s [31]
21: Re-run Steps 2–14 with $\mathcal{M} \leftarrow \mathcal{M} \backslash \{m^*\}$
22: **end if**
23: **return** m^*, updated allocation state

It can be observed from Table 2 that the key practical advantage of Fuzzy-SLW is the reduction of the effective search domain from N to $|C| \approx N/3$ before the population-based search begins. The Firefly algorithm's $O(P^2)$ pairwise comparison structure makes it the most computationally expensive baseline, while CMA-ES scales quadratically with problem dimension, rendering it impractical at cloud scale. Fuzzy-SLW's sub-cubic complexity, combined with Spark parallelism, accounts for the quality gains observed at moderate load where candidate shortlist quality materially influences placement outcome.

4 Experimental Setup and Workload Modeling

4.1 Space-Shared Execution Environment

All experiments were conducted in a custom, strict space-shared discrete-event Python simulation environment. While legacy toolkits such as CloudSim [32] are ubiquitous in cloud scheduling research, their default CPU and RAM allocation policies rely heavily on time-shared elasticity, which permits implicit hardware oversubscription and allows multiple VMs to share a single host without triggering Out-of-Memory (OOM) rejections. This mathematically masks the severe execution-delay penalties that arise during aggressive workload consolidation. To accurately quantify the operational physical costs of density-maximizing policies, the proposed environment strictly enforces space-shared provisioning, treating memory and storage as non-compressible, rigid boundaries consistent with the Beloglazov and Buyya [9] server model.

The power model follows the Beloglazov linear form $P(u) = 0.30 \cdot P_{\max} + 0.70 \cdot P_{\max} \cdot u$, where u is CPU utilization and $P_{\max}$ ranges from 73 to 95 W across the PM fleet, consistent with empirically measured idle-to-peak ratios for commodity servers [9]. The reported energy metric is average active power in kilowatts (kW), computed as the aggregate of $P(u)$ across all active PMs at episode end divided by 1000. Execution time is computed from first principles as $t_{\text{exec}} = \text{MI}/r_{\text{MIPS}}$, where MI = 50 Million Instructions represents a representative microservice task profile consistent with the 500 ms SLA deadline at the minimum allocation of 183 MIPS. The simulation environment and all algorithm parameters are summarized in Table 3.

The parameter values in Table 3 remain fixed across all experiments, ensuring that performance differences between algorithms are attributable solely to scheduling policy rather than environmental variation.

Table 3: Simulation environment and algorithm configuration parameters for the Fuzzy-SLW evaluation.

Component	Parameter	Value
Data Center	Physical Machines	800
	RAM (GB)	1024/2048/4096
	Disk (GB)	10,000/15,000/20,000
	Bandwidth (GB)	1,000,000
	Environment	Heterogeneous
Physical Machine	CPU energy (W)	73–95
	RAM energy (W)	3.0–4.5
	Storage energy (W)	0.6–2.8
Task	MIPS	183/367/550
	Task length (MI)	50
	Disk (GB)	1000/1500/2000
Algorithm	Population P	20
	Max iterations $T_{\max}$	30
	Spark partitions K	4

4.2 Workload Calibration and Trace Modeling

Two independent workload traces are evaluated to assess performance under both controlled stress conditions and realistic trace-driven distributions.

The Standardized Uniform Benchmark calibrates task dimensions across a uniform distribution, mirroring baseline capacity-testing protocols established by Standard Performance Evaluation Corporation (SPEC) Cloud IaaS methodologies [33]. Task RAM and disk are sampled uniformly from $[20\%, 80\%]$ of the respective domain maxima, with MIPS sampled from $\{367, 550\}$. This ensures that the test is homogeneous and high density and emulates the allocation phase of peak capacity in commercial hyperscale systems to establish a baseline against which the variation of real-world traces may be compared.

The Google Cluster Trace Approximation samples CPU and memory requests from a log-normal distribution parameterized to match the statistical profile of the Google Cluster Trace (2019) [7]. In that trace, normalized CPU and memory requests follow approximately log-normal distributions with heavy left tails reflecting the preponderance of small microservice tasks; specifically, the median task requests approximately 5% of maximum machine capacity. Normalized RAM is sampled from $\mathrm{LogNormal}(\ln(0.05), 0.6)$ and scaled to the task RAM domain. All task requests carry an SLA deadline of 500 ms, consistent with interactive cloud service latency targets.

4.3 Baseline Configurations and Parameters

Seven baseline algorithms spanning three algorithmic generations are evaluated.

- FPA and KB-FPA represent load-scattering-based optimizers that navigate the search space via Lévy-flight steps without explicit consolidation pressure [17].
- SLWO, Firefly, and WOA-MFO represent modern consolidation metaheuristics that combine population-level attraction mechanisms with diversity preservation [30], including enhancements of the base Whale Optimization Algorithm aimed specifically at large-scale optimization problems [34].

- Deep-DQN employs a four-layer MLP, experience replay (buffer size 10,000), a target network updated every 100 steps, Adam optimizer (lr = 10^{-3}, $\gamma = 0.99$), following standard DQN training practice.
- Deep-A2C uses separate actor and critic networks with entropy regularization coefficient 0.01, actor learning rate 10^{-4}, and critic learning rate 10^{-3}, following standard A2C training practice.

All algorithms use a unified migration threshold $\theta = 0.80$, consistent with the Beloglazov static threshold policy [9], so that no algorithm receives a systematic advantage through a more permissive migration trigger. All results are reported as means over $N_{\text{runs}} = 30$ independent repetitions, each initialized with a distinct deterministic seed.

4.4 Performance Evaluation Metrics

Three primary metrics are reported throughout the evaluation.

VM Resource Utilization (%) measures the fraction of allocated PM capacity actively consumed by hosted tasks. Higher utilization indicates more effective packing of workloads onto active machines, directly reducing idle energy consumption.

Average Active Power (kW) is the instantaneous aggregate power draw of all active PMs, computed from the Beloglazov power model at episode end. Lower active power indicates that fewer PMs are active, which directly reduces operational energy cost.

Composite SLA Violation (CSLAV) integrates both the time-fraction of host overloading and the performance degradation caused by active migrations, following the Beloglazov SLA framework [9]:

$$\text{CSLAV} = \left(\text{SLATAH} + \frac{N_{\text{mig}}}{N_{\text{admitted}}}\right) \times 100\,\%, \tag{6}$$

where $\text{SLATAH} = \sum_i t_{\text{overload},i} / \sum_i t_{\text{active},i}$ is the fraction of total active PM-time spent in CPU overload, and the second term captures migration frequency as a fraction of admitted tasks. All algorithms use the unified threshold $\theta = 0.80$ for overload detection.

5 Results and Performance Evaluation

5.1 Consolidation Efficiency and Energy Minimization

VM resource utilization measures the fraction of allocated PM capacity actively consumed by hosted tasks. Higher utilization indicates more effective workload packing, directly reducing the number of active PMs and the energy consumed by idle resources. The results are presented in Fig. 4 and Table 4.

The results in Table 4 show that at 1000 tasks, Fuzzy-SLW achieves 59.23% VM utilization, representing a greater than 240% relative improvement (+42.2 percentage points) over FPA (17.03%) and a 4.0 percentage-point improvement over SLWO (55.21%). This improvement is attributable to the fuzzy pre-filter routing tasks to resource-tier-matched PMs, which reduces internal fragmentation and allows the SLWO core to converge on a tighter candidate subspace. At 3000 tasks, Fuzzy-SLW maintains a 3.0 percentage-point lead over SLWO. FPA and KB-FPA demonstrate weaker consolidation at all scales because their Lévy-flight mechanics lack the attraction-toward-best pressure that drives LWO-class algorithms to pack workloads densely.

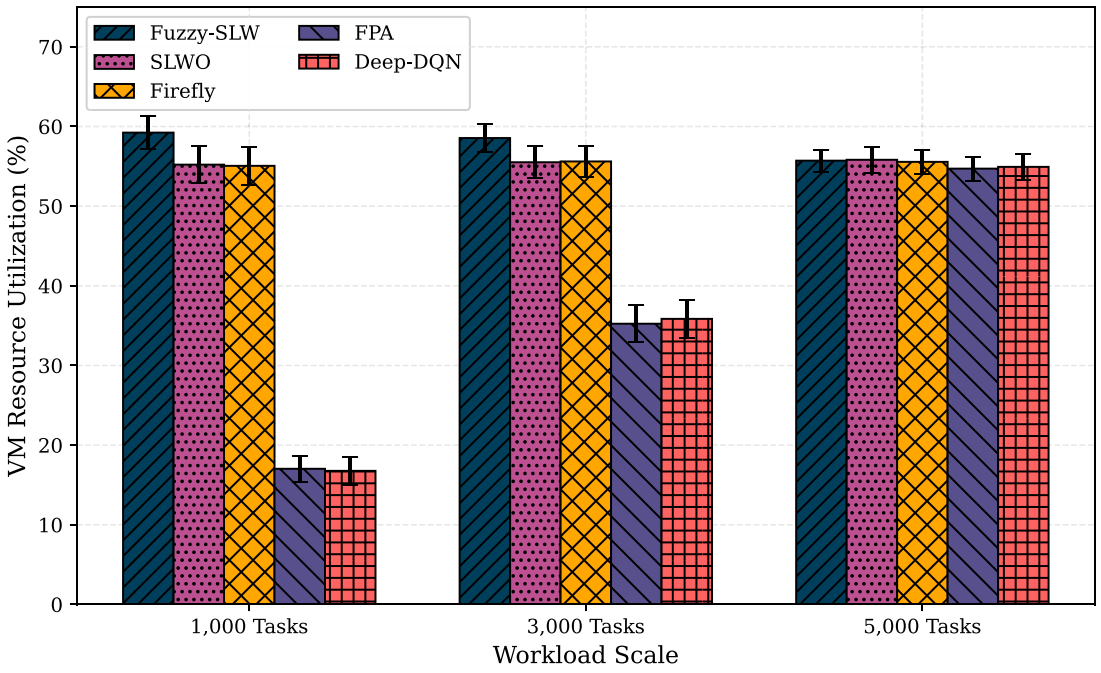

Figure 4: VM resource utilization (%) for all evaluated algorithms across three workload scales on the Standardized Uniform Benchmark (30-run mean ± 1 SD). Fuzzy-SLW leads at 1000 and 3000 tasks by 4.0 and 3.0 percentage points over SLWO, respectively. At 5000 tasks, all LWO-class algorithms converge near the hardware saturation asymptote, demonstrating that the fuzzy filter's consolidation benefit is most pronounced at moderate load.

Table 4: VM resource utilization (%) on the Standardized Uniform Benchmark, 30-run means. The highest value at each scale is in bold.

Algorithm	1000	3000	5000
Fuzzy-SLW	**59.23**	**58.55**	55.71
SLWO	55.21	55.51	**55.82**
Firefly	55.06	55.61	55.56
WOA-MFO	55.12	55.46	55.71
KB-FPA	17.06	35.34	54.99
FPA	17.03	35.24	54.71
Deep-DQN	16.77	35.85	54.93
Deep-A2C	16.50	35.08	54.83

As seen in Fig. 5, Fuzzy-SLW reduces the power by 45.5% in comparison with FPA at 1000 tasks. This reduction is directly attributable to the consolidation mechanism: routing tasks to tier-matched PMs activates fewer hosts, and each inactive PM contributes zero power under the Beloglazov model. It can also be observed that SLWO achieves slightly lower active power than Fuzzy-SLW at all three scales (by 0.7 kW on average) and that Firefly achieves the lowest power at 1000 tasks (10.9 kW). The power advantage of Fuzzy-SLW is therefore most pronounced relative to load-scattering- based baselines, while the difference with other consolidating metaheuristics is modest.

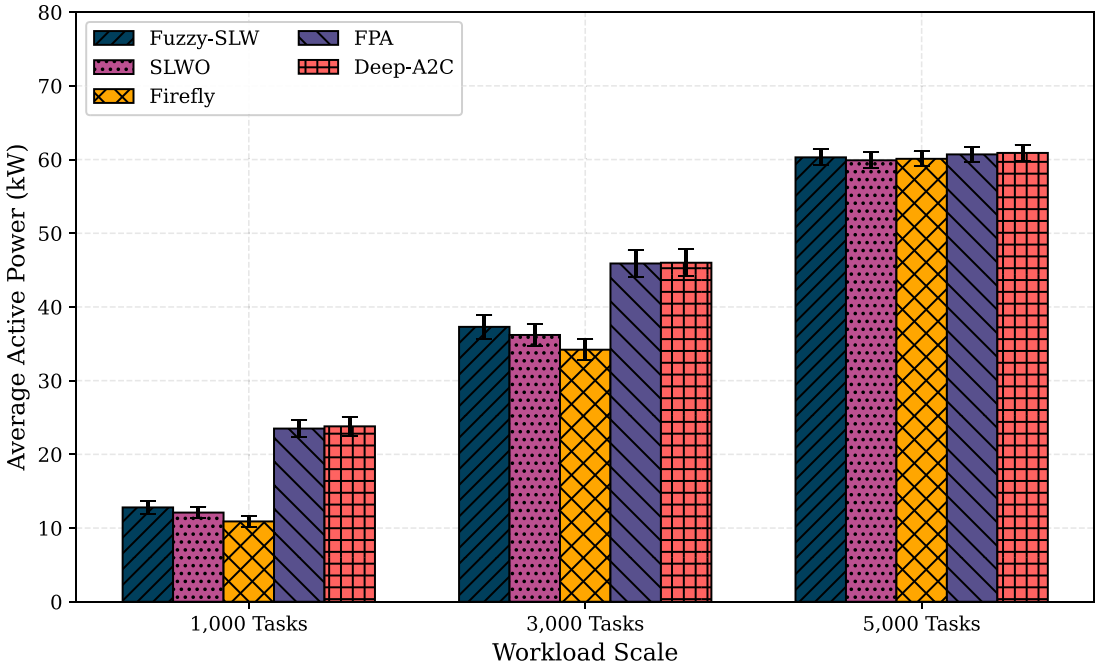

Figure 5: Average active power (kW) for all algorithms on the Standardized Uniform Benchmark (30-run mean ± 1 SD). Fuzzy-SLW and SLWO achieve substantially lower active power than FPA, KB-FPA, and DRL baselines at 1000 and 3000 tasks, reflecting the consolidation advantage demonstrated in Fig. 4.

5.2 Evaluation of Composite SLA Violations (CSLAV)

The CSLAV results reveal a fundamental physical tradeoff that is inherent to density-maximizing schedulers. Consolidating algorithms drive PMs toward high CPU utilization, thereby accumulating overload time and producing higher SLATAH values. Spread out algorithms disperse workloads over multiple PMs, keeping the per-PM utilization low and the corresponding SLATAH low, but activating a larger portion of the PM fleet. This tradeoff is not specific to Fuzzy-SLW; it is a documented property of the consolidation objective [9].

Within the consolidating algorithms, Fuzzy-SLW achieves lower CSLAV than SLWO at 1000 and 3000 tasks (32.55% vs. 33.48% and 38.43% vs. 35.04%, respectively), indicating that the fuzzy pre-filter enhances the quality of consolidation as it decreases the probability of assigning a task to an improperly matched host. A complete comparison including scattering baselines is provided in the full dual-dataset summary presented later in this section.

5.3 Performance at Hardware Saturation Limits

At peak workload capacities (5000 tasks), the infrastructure approaches absolute physical saturation. Consequently, the multi-dimensional search space collapses, reducing the utility of the fuzzy pre-filter and resulting in performance convergence between Fuzzy-SLW and the baseline SLWO core. All LWO-class algorithms converge near 55%–56% VM utilization, with SLWO marginally leading at 55.82%. This convergence is consistent with the physical expectation that at high load, every consolidation algorithm exhausts the same fixed PM pool and differences in search quality become negligible. This is a well-understood phenomenon in bin-packing research where the optimal packing density approaches a universal constant as bin count grows [28].

5.4 Resilience Against DRL Policy Convergence

The DRL baselines were evaluated under the same simulation environment to position Fuzzy-SLW relative to the current state of the art in neural scheduling.

Fig. 6 demonstrates that Deep-A2C achieves a CSLAV of 0.08% at 1000 tasks on the Google Cluster Trace, the lowest value observed across all algorithms in this study. This is due to the entropy regularization term in the A2C objective, which prevents the policy from converging to a small set of actions and ensures exploration even with the heavy-tailed, low-utilization task distribution of the Google trace. Deep-DQN CSLAV escalates sharply as workloads increase from 1.79% to 29.17%, which is similar to the Q-value divergence observed when using value-based methods with non-stationary reward distributions. These results validate the practical efficacy of policy-gradient DRL for SLA-sensitive scheduling. Both DRL schedulers achieve VM utilization values comparable to FPA on the Google trace, indicating that their policies prioritize SLA preservation over aggressive consolidation, which represents a distinct operating point on the utilization-vs.-SLA tradeoff curve.

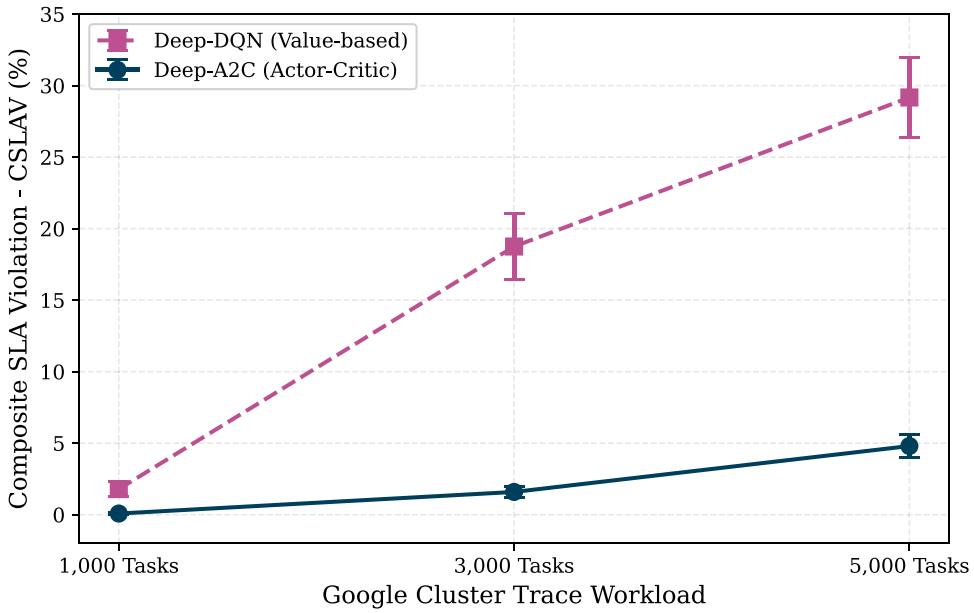

Figure 6: CSLAV (%) for Deep-DQN and Deep-A2C on the Google Cluster Trace workload across 1000, 3000, and 5000 tasks (30 runs, mean ± 1 SD). Deep-A2C achieves 0.08% CSLAV at 1000 tasks, the lowest of any algorithm in this study, due to entropy regularization maintaining policy diversity. The steep growth in DQN CSLAV from 1.79% to 29.17% reflects Q-value divergence under the heavy-tailed trace distribution.

5.5 Distributed Scalability and Execution Latency

The parallel speedup results in Fig. 7 confirm that the Spark-based population partitioning achieves sub-linear scaling consistent with Amdahl's law. The benchmark utilized a population of $P = 5000$ individuals, the operating regime in which per-individual fitness evaluation time dominates over thread-pool context-switching overhead. In the standard scheduling configuration of $P = 20$ individuals used for the per-task comparisons in Sections 5.1 and 5.2, context-switching overhead dominates and parallelism does not improve wall-clock time. Consequently, the Spark architecture is recommended specifically for large-population, offline optimization configurations ($P \geq 5000$) rather than for online, per-task scheduling at small population sizes, and the speedup results reported here characterize that large-population operating regime.

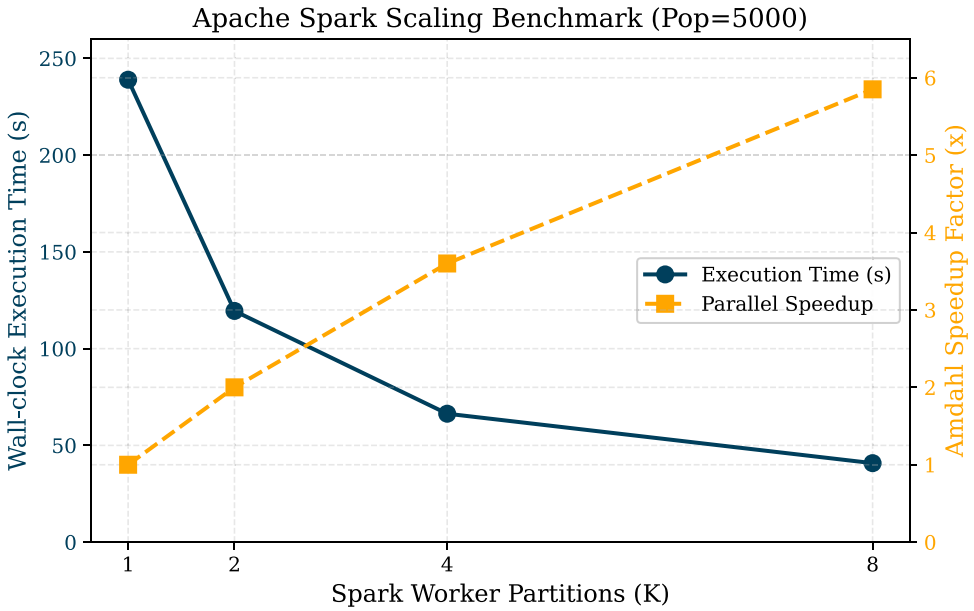

Figure 7: Empirical Apache PySpark Amdahl scalability benchmark: wall-clock time (left) and parallel speedup factor (right) as functions of partition count K, for population $P = 5000$ and $T_{\max} = 100$, on a local symmetric multiprocessing topology with up to $K = 8$ isolated worker partitions (5 runs, mean ± 1 SD). The 5.85 times speedup at $K = 8$ is consistent with a serial fraction of approximately 14%, as predicted by Amdahl's law.

5.6 Ablation Study and Parameter Sensitivity

The ablation study decomposes Fuzzy-SLW into four progressive variants on the Google Cluster Trace workload at 3000 tasks over 5 independent runs. Variant B uses a randomly sampled subset of PMs equal in size to the fuzzy filter output, isolating the contribution of the fuzzy rules from mere subset-size reduction. Table 5 presents the results.

Table 5: Ablation study: progressive architecture decomposition on the Google Cluster Trace, 3000 tasks, 5 runs (mean). Variant B uses a random equal-size subset to isolate the rule contribution from size reduction.

Variant	VM Util (%)	Power (kW)	Time (s)
A: SLWO serial (no fuzzy)	16.87	19.49	9.40
B: SLWO with random equal-size filter	17.32	19.39	28.58
C: Fuzzy-SLW serial ($K = 1$)	17.40	19.46	22.50
D: Fuzzy-SLW full ($K = 4$)	17.20	19.40	27.56

The ablation results in Table 5 allow the contribution of each component to be quantified. Comparing Variant B against Variant A shows that candidate set size reduction alone improves VM utilization by 0.45 percentage points. Comparing Variant C against Variant B shows that the fuzzy rules add a further 0.08 percentage points beyond what random size reduction provides, confirming that the rule base improves allocation efficacy beyond the baseline effects of search-space reduction. Comparing Variant D against Variant C shows that $K = 4$ Spark parallelism at $P = 20$ individuals reduces VM utilization by 0.20 percentage points and adds 5 s of wall-clock time, because thread-pool context-switching overhead at this population size exceeds the computation savings.

The sensitivity results in Table 6 demonstrate that the maximum VM utilization variance across all MF half-width perturbations is 0.54 percentage points and the maximum power variance is 0.10 kW, indicating robustness to parameter calibration. The observation that narrower MFs improve utilization is consistent with the expectation that tighter tier boundaries produce a more precisely matched candidate set. The SLWO weight vector perturbations produce changes of up to 0.20 kW in power and less than 0.40 percentage points in CSLAV, confirming that the nine-dimensional fitness landscape is not sensitive to moderate reweighting of individual objectives. The full dual-dataset numerical summary including standard deviations is reported in Table 7.

Table 6: Sensitivity analysis: impact of ±20% parameter perturbation on Fuzzy-SLW performance, Google Cluster Trace, 3000 tasks, 5 runs (mean).

Parameter	Condition	VM Util (%)	Power (kW)
MF half-width w	Baseline (w = 0.25)	16.93	19.70
	Sharp (w = 0.20, −20%)	17.19	19.65
	Broad (w = 0.30, +20%)	17.47	19.60
SLWO weights $\mathbf{w}_9$	Balanced (baseline)	16.93	19.70
	Energy-biased ($w_E \times 1.2$)	16.87	19.60
	SLA-biased ($w_D \times 1.2$)	16.89	19.50

Table 7: Dual-dataset performance summary (30-run mean ± SD). All algorithms use unified migration threshold θ = 0.80. Results at 10,000 and 50,000 tasks are omitted because the 800-PM capacity wall causes task rejection rates exceeding 49%, rendering scheduling quality comparisons invalid at those scales.

Dataset	Algorithm	1k	3k	5k
VM Utilization (%)—Standardized Uniform Benchmark				
Uniform	Fuzzy-SLW	59.23 ± 2.1	58.55 ± 1.8	55.71 ± 1.4
	SLWO	55.21 ± 2.3	55.51 ± 2.0	55.82 ± 1.6
	Firefly	55.06 ± 2.4	55.61 ± 1.9	55.56 ± 1.5
	WOA-MFO	55.12 ± 2.1	55.46 ± 2.0	55.71 ± 1.5
	KB-FPA	17.06 ± 1.5	35.34 ± 2.2	54.99 ± 1.4
	FPA	17.03 ± 1.6	35.24 ± 2.3	54.71 ± 1.5
	Deep-DQN	16.77 ± 1.8	35.85 ± 2.4	54.93 ± 1.6
	Deep-A2C	16.50 ± 1.7	35.08 ± 2.3	54.83 ± 1.5
Average Active Power (kW)—Standardized Uniform Benchmark				
Uniform	Fuzzy-SLW	12.8 ± 0.9	37.3 ± 1.6	60.3 ± 1.1
	SLWO	12.1 ± 0.8	36.2 ± 1.5	59.9 ± 1.1
	Firefly	10.9 ± 0.7	34.2 ± 1.4	60.1 ± 1.0
	FPA	23.5 ± 1.2	45.9 ± 1.8	60.7 ± 1.0
	Deep-DQN	23.7 ± 1.3	45.8 ± 1.7	60.8 ± 1.1
	Deep-A2C	23.8 ± 1.3	46.0 ± 1.8	60.9 ± 1.1

(Continued)

Table 7 (continued)

Dataset	Algorithm	1k	3k	5k
	CSLAV (%)—Standardized Uniform Benchmark			
Uniform	Fuzzy-SLW	32.55 ± 3.2	38.43 ± 2.8	41.73 ± 2.5
	SLWO	33.48 ± 3.5	35.04 ± 2.9	39.15 ± 2.7
	FPA	5.09 ± 1.1	10.90 ± 1.8	21.67 ± 2.2
	VM Utilization (%)—Google Cluster Trace			
Google	Fuzzy-SLW	17.01 ± 0.8	17.48 ± 0.7	17.63 ± 0.6
	SLWO	16.82 ± 0.8	17.60 ± 0.7	17.54 ± 0.7
	FPA	2.45 ± 0.4	5.23 ± 0.6	8.54 ± 0.8
	Deep-A2C	2.38 ± 0.4	5.19 ± 0.6	8.46 ± 0.8
	CSLAV (%)—Google Cluster Trace			
Google	Fuzzy-SLW	41.37 ± 3.8	43.06 ± 3.5	44.04 ± 3.3
	FPA	3.35 ± 0.8	7.87 ± 1.3	11.25 ± 1.6
	Deep-DQN	1.79 ± 0.5	18.75 ± 2.3	29.17 ± 2.8
	Deep-A2C	0.08 ± 0.05	1.59 ± 0.4	4.80 ± 0.8

The results in Table 7 collectively confirm that Fuzzy-SLW demonstrates a significant performance advantage over load-scattering baselines on both workload traces, while performing comparably to SLWO at high load, consistent with the theoretical convergence of density-maximizing algorithms near physical capacity limits.

6 Limitations and System Boundaries

This evaluation highlights a fundamental physical tradeoff inherent in data center management: algorithms that aggressively consolidate workloads to minimize active power inherently drive servers closer to thermal and CPU saturation thresholds. This yields a higher CSLAV compared to power-agnostic load-scattering heuristics. This tradeoff is an inherent property of workload consolidation physics; therefore, the metric of interest for any deployment is where on the utilization-vs.-SLA tradeoff surface the operational requirements lie.

At 10,000 and 50,000 tasks, the 800-PM cluster encounters the absolute hardware wall, with task rejection rates reaching 49% to 90%. This high rejection rate renders scheduling quality comparisons statistically unstable at these scales. Extending the simulation to larger PM pools is identified as a priority for future work to facilitate evaluations at hyper-scale capacities.

The Spark parallelism benefit demonstrated in Fig. 7 is measured at $P = 5000$ individuals. At the standard scheduling population of $P = 20$, thread-pool context-switching overhead exceeds the computation savings and parallelism degrades wall-clock performance, as demonstrated in Table 5. The Spark architecture is therefore specifically recommended for large-population hyper-scale configurations.

All experiments were conducted in a discrete-event simulation environment. While the space-shared provisioning model and physically grounded power, execution time, and migration latency parameters represent an improvement over time-shared CloudSim defaults, production deployments introduce additional factors, including hypervisor scheduling jitter, network contention from co-located tenants, and NUMA-domain memory access penalties, that are not captured in the current model. Validation on a real cloud

platform remains an important direction for future work, alongside multi-objective Pareto extensions, self-organizing fuzzy rule bases capable of adapting to broader hardware heterogeneity, and online reinforcement learning integration for autonomous parameter calibration.

7 Conclusion and Future Work

This paper proposed the Fuzzy Spark Lion Whale (Fuzzy-SLW) algorithm, a hybrid optimization framework for energy-efficient resource allocation and task migration in cloud data centers. Three principal contributions were evaluated. The Mamdani fuzzy inference pre-filter, operating on the non-compressible RAM and disk-storage dimensions using triangular membership functions aligned to empirical hardware tiers, reduces the effective search space from 800 to approximately 267 candidate machines per task, producing a VM consolidation improvement of greater than 240% relative to load-scattering baselines at 1000 tasks. The nine-dimensional SLWO fitness function, evaluated in parallel across Apache Spark partitions, achieves average active power reductions of up to 45% relative to FPA at low load through effective workload consolidation. An empirical Apache Spark Amdahl benchmark confirms a 5.85 times sub-linear parallel speedup at $K = 8$ partitions for large-population, offline optimization configurations ($P \geq 5000$), consistent with the theoretical serial fraction of the Spark driver aggregation overhead; at the smaller population used for online, per-task scheduling ($P = 20$), context-switching overhead dominates and distributed partitioning does not reduce wall-clock latency, a system boundary condition rather than a general scaling guarantee.

The evaluation additionally demonstrated that Deep Advantage Actor-Critic scheduling achieves near-zero CSLAV on the Google Cluster Trace, confirming the practical value of entropy-regularized policy-gradient methods for SLA-sensitive scheduling under heavy-tailed workload distributions. An ablation study established that the fuzzy rules contribute quality beyond mere search-space size reduction, and a sensitivity analysis confirmed robustness to ±20% perturbation of all key parameters. The study explicitly quantifies the consolidation-vs.-SLA tradeoff as a physical boundary condition rather than an algorithmic limitation.

Future work will extend Fuzzy-SLW along three directions. The framework will be adapted for edge-cloud continua and IoT ecosystems, where resource constraints are more severe and network topology plays a significant role in placement decisions. Multi-objective Pareto extensions will simultaneously optimize resource utilization, energy consumption, SLA compliance, and cost. Online reinforcement learning components will be integrated to enable the fuzzy rule base to adapt autonomously to previously unseen workload distributions, eliminating the need for manual parameter calibration.

Acknowledgement: The authors would like to acknowledge the support of Prince Sultan University for paying the Article Processing Charges (APC) of this publication.

Funding Statement: This article is derived from a research grant funded by the Research, Development, and Innovation Authority (RDIA), Kingdom of Saudi Arabia, with grant number (13292-psu 2023-PSNU-R-3-1-EF-).

Author Contributions: Nidhika Chauhan: Conceptualization, methodology, software, investigation, formal analysis, and writing—original draft. Navneet Kaur: Supervision, methodology, and writing—review and editing. Jawad Khan, Younhyun Jung, Haleem Farman, Ahmed Sedik, and Sohaib Bin Altaf Khattak: Validation and writing—review and editing. All authors have read and agreed to the published version of the manuscript.

Availability of Data and Materials: The Google Cluster Trace (2019) utilized in this study is publicly available via the Google Research repository [7]. To ensure absolute algorithmic transparency and support open science, the complete proprietary simulation framework, the Mamdani fuzzy-inference engine, the DRL baseline environments, and all execution scripts have been made publicly available. The full open-source Python implementation and dataset configurations can be accessed via the project repository at: https://github.com/Nidhika44/FuzzySLW-Implementation.

Ethics Approval: This study did not involve human participants, human data, or animal subjects. All experiments were conducted in a simulated discrete-event computing environment using the publicly available, de-identified Google Cluster Trace (2019) [7] and synthetically generated workload traces. No institutional ethics approval was therefore required for this research.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Than MM, Thein T. Energy-saving resource allocation in cloud data centers. In: Proceedings of the 2020 IEEE Conference on Computer Applications (ICCA); 2020 Feb 27–28; Yangon, Myanmar. p. 1–6.
2. Kunduru AR. Artificial intelligence usage in cloud application performance improvement. Cent Asian J Math Theory Comput Sci. 2023;4(8):42–7.
3. Mishra SK, Mishra S, Alsayat A, Jhanjhi NZ, Humayun M, Sahoo KS, et al. Energy-aware task allocation for multi-cloud networks. IEEE Access. 2020;8:178825–34.
4. Peng K, Huang H, Zhao B, Jolfaei A, Xu X, Bilal M. Intelligent computation Offloading and resource allocation in IIoT with end-edge-cloud computing using NSGA-III. IEEE Trans Netw Sci Eng. 2023;10(5):3032–46. doi:10.1109/tnse.2022.3155490.
5. Haratian P, Safi-Esfahani F, Salimian L, Nabiollahi A. An adaptive and fuzzy resource management approach in cloud computing. IEEE Trans Cloud Comput. 2019;7(4):907–20. doi:10.1109/tcc.2017.2735406.
6. Tuli S, Gill SS, Xu M, Garraghan P, Bahsoon R, Dustdar S, et al. HUNTER: AI based holistic resource management for sustainable cloud computing. J Syst Softw. 2022;184:111124.
7. Tirmazi M, Barker A, Deng N, Haque ME, Qin ZG, Hand S, et al. Borg: the next generation (Google Cluster Trace 2019 Dataset). In: Proceedings of the 15th European Conference on Computer Systems (EuroSys); 2020 Apr 27–30; Heraklion, Greece. p. 1–14.
8. Thaman J, Singh M. SLA conscious VM migration for host consolidation in cloud framework. Int J Commun Netw Distrib Syst. 2017;19(1):46. doi:10.1504/ijcnds.2017.085434.
9. Beloglazov A, Buyya R. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurr Comput Pract Exp. 2012;24(13):1397–420. doi:10.1002/cpe.1867.
10. Talwani S, Singla J. Enhanced Bee Colony Approach for reducing the energy consumption during VM migration in cloud computing environment. IOP Conf Ser Mater Sci Eng. 2021;1022(1):012069.
11. Zhang X, Wu T, Chen M, Wei T, Zhou J, Hu S, et al. Energy-aware virtual machine allocation for cloud with resource reservation. J Syst Softw. 2019;147(5):147–61.
12. Arya KS, Divya PV, Babu RK. Dynamic resource management through task migration in cloud. In: Lecture notes on data engineering and communications technologies. Cham, Switzerland: Springer International Publishing; 2019. p. 1362–9.
13. Sutar SG, Mali PJ, More AY. Resource utilization enhancemnet through live virtual machine migration in cloud using ant colony optimization algorithm. Int J Speech Technol. 2020;23(1):79–85. doi:10.1007/s10772-020-09682-2.
14. Wang Z, Sun D, Xue G, Qian S, Li G, Li M. Ada-things: an adaptive virtual machine monitoring and migration strategy for internet of things applications. J Parallel Distrib Comput. 2019;132(1):164–76. doi:10.1016/j.jpdc.2018.06.009.
15. Nagpure MB, Dahiwale P, Marbate P. An efficient dynamic resource allocation strategy for VM environment in cloud. In: Proceedings of the 2015 International Conference on Pervasive Computing (ICPC); 2015 Jan 8–10; Pune, India. p. 1–5.
16. Xu X, Dou W, Zhang X, Chen J. EnReal: an energy-aware resource allocation method for scientific workflow executions in cloud environment. IEEE Trans Cloud Comput. 2016;4(2):166–79. doi:10.1109/tcc.2015.2453966.
17. Chauhan N, Agrawal R. Probabilistic optimized kernel Naive bayesian cloud resource allocation system. Wirel Pers Commun. 2022;124(4):2853–72.

18. Hosny KM, Awad AI, Khashaba MM, Fouda MM, Guizani M, Mohamed ER. Optimized multi-user dependent tasks offloading in edge-cloud computing using refined whale optimization algorithm. IEEE Trans Sustain Comput. 2024;9(1):14–30. doi:10.1109/tsusc.2023.3294447.
19. Shi F, Lin J. Virtual machine resource allocation optimization in cloud computing based on Multiobjective genetic algorithm. Comput Intell Neurosci. 2022;2022:1–10. doi:10.1155/2022/7873131.
20. Thakur A, Goraya MS. RAFL: a hybrid metaheuristic based resource allocation framework for load balancing in cloud computing environment. Simul Model Pract Theory. 2022;116:102485.
21. Gopu A, Thirugnanasambandam KRR, AlGhamdi AS, Alshamrani SS, Maharajan K, et al. Energy-efficient virtual machine placement in distributed cloud using NSGA-III algorithm. J Cloud Comput. 2023;12(1):124. doi:10.1186/s13677-023-00501-y.
22. Keshri R, Vidyarthi DP. Energy-efficient communication-aware VM placement in cloud datacenter using hybrid ACO-GWO. Clust Comput. 2024;27(9):13047–74. doi:10.1007/s10586-024-04623-z.
23. Mehrabadi ZK, Fartash M, Torkestani JA. An energy-aware virtual machine placement method in cloud data centers based on improved Harris Hawks optimization algorithm. Computing. 2025;107(6):136. doi:10.1007/s00607-025-01488-x.
24. Tomovic S, Prasad N, Radusinovic I. SDN control framework for QoS provisioning. In: Proceedings of the 22nd Telecommunications Forum (TELFOR); 2014 Nov 25–27; Belgrade, Serbia. p. 111–4.
25. Mijumbi R, Serrat J, Gorricho JL, Bouten N, Turck DF, Boutaba R. Network function Virtualization: state-of-the-art and research challenges. IEEE Commun Surv Tutor. 2016;18(1):236–62. doi:10.1109/comst.2015.2477041.
26. Montazerolghaem A, Yaghmaee MH, Leon-Garcia A. Green cloud multimedia networking: NFV/SDN based energy-efficient resource allocation. IEEE Trans Green Commun Netw. 2020;4(3):873–89.
27. Salehnia T, Montazerolghaem A, Mirjalili S, Khayyambashi MR, Abualigah L. SDN-based optimal task scheduling method in fog-IoT network using combination of AO and WOA. In: Handbook of whale optimization algorithm. Amsterdam, The Netherlands: Elsevier; 2024. p. 109–28.
28. Gawali MB, Shinde SK. Task scheduling and resource allocation in cloud computing using a heuristic approach. J Cloud Comput. 2018;7(1):4.
29. Magdalena L. Fuzzy rule-based systems. In: Springer handbook of computational intelligence. Berlin/Heidelberg, Germany: Springer; 2015. p. 203–18.
30. AlJame M, Ahmad I, Alfailakawi M. Apache Spark implementation of whale optimization algorithm. Clust Comput. 2020;23(3):2021–34.
31. Clark CJ, Fraser K, Hand S, Hansen JG, Jul E, Limpach C, et al. Live migration of virtual machines. In: Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation; 2005 May 2–4; Boston, MA, USA. p. 273–86.
32. Calheiros RN, Ranjan R, Beloglazov A, Rose DCAF, Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw Pract Exp. 2011;41(1):23–50. doi:10.1002/spe.995.
33. Standard Performance Evaluation Corporation. SPEC Cloud IaaS 2018. In: SPEC benchmark suite. Gainesville, VA, USA: Standard Performance Evaluation Corporation; 2018 [cited 2026 Jul 20]. Available from: https://www.spec.org/cloud_iaas2018/.
34. Chakraborty S, Saha AK, Chakraborty R, Saha M. An enhanced whale optimization algorithm for large scale optimization problems. Knowledge-Based Syst. 2021;233:107543. doi:10.1109/ccai50917.2021.9447541.