National Research Journal of Information Technology & Information Science ISSN: 2350-1278
Volume-13, Issue-1, Year-2026 (January-June) Peer Reviewed & Refereed Journal (IF: 7.9)
PP: 471-495 Journal Website www.nrjitis.in

ADAPTIVE PARTICLE SWARM OPTIMIZATION FOR RESOURCE-EFFICIENT CLOUD TASK SCHEDULING: A CONVERGENCE-AWARE FRAMEWORK

Harshita Shukla

Research Scholar
Department of Computer Science and Engineering, Apex University, Jaipur, Rajasthan, India

Vijay Singh Rathore

Professor
Computer Science and Engineering & Dean International, Apex University, Jaipur, Rajasthan, India

ABSTRACT

Cloud task scheduling requires a scheduler to map heterogeneous computational tasks to virtual machines while balancing service time, monetary cost, and infrastructure utilization. Particle swarm optimization (PSO) offers a compact population-based search mechanism for this combinatorial problem, yet conventional PSO implementations are sensitive to parameter selection, may converge prematurely, and commonly rely on fixed iteration budgets or isolated stall counters. This paper presents a modular convergence-aware adaptive PSO framework for resource-efficient scheduling of independent cloud tasks. The framework separates the cloud model, task-to-virtual-machine encoding, fitness composition, parameter adaptation, convergence monitoring, recovery, and termination logic so that individual mechanisms can be replaced or extended without redesigning the complete scheduler. A MATLAB-only implementation was developed and verified through unit tests and pilot experiments. In the 40-task and 6-VM demonstration, Max-Min, standard PSO, and the adaptive convergence-aware PSO produced the same best observed mean fitness of 0.12561, with makespan 29.159 and utilization 0.98229. In the 100-task and 10-VM experiment, standard PSO achieved the lowest observed mean fitness of 0.054792, slightly improving over Max-Min and the current adaptive configuration, both of which obtained 0.055130. The results verify the proposed software architecture and confirm the value of resource-aware scheduling against round-robin allocation. However, the current adaptive configuration did not yet demonstrate evaluation savings or superiority over standard PSO. The paper therefore reports the framework as a verified and extensible research platform, while identifying convergence-threshold tuning, recovery calibration, and large-scale repeated experiments as necessary future work.

Keywords—adaptive particle swarm optimization, cloud computing, cloud task scheduling, convergence detection, resource utilization, makespan, scheduling cost, autonomous termination

I. INTRODUCTION

Cloud computing has evolved into a service-oriented computational model in which processing capacity, storage, platforms, and software are provisioned through virtualized infrastructure. The abstraction of physical hardware enables rapid scaling, but it also transfers a substantial operational responsibility to the cloud scheduler. When many users submit tasks with different computational requirements, the scheduler must decide which virtual machine (VM) should execute each task and in what order. A poor allocation may leave some VMs idle while others remain overloaded, increase the completion time of the workload, raise the

execution charge, and reduce the quality of service experienced by users. Task scheduling is therefore not an auxiliary implementation detail; it is a central optimization problem that influences both provider efficiency and user-level performance.

For a set of independent tasks and heterogeneous VMs, the number of feasible mappings grows exponentially with the size of the problem. Exact enumeration rapidly becomes impractical, and the scheduler must often produce a high-quality solution within a limited decision time. Heuristics such as first-come-first-served, round robin, minimum completion time, Min-Min, and Max-Min are inexpensive and interpretable, but their greedy decisions may not adequately explore trade-offs among makespan, cost, utilization, and load distribution. Metaheuristic methods have consequently been investigated because they can search large scheduling spaces without requiring differentiability or a convex mathematical model. Among these methods, particle swarm optimization (PSO) is attractive because its update mechanism is compact, the number of control parameters is small, and candidate schedules exchange information through personal and global experience [1].

The original PSO was formulated for continuous spaces. Cloud task scheduling, however, is a discrete assignment problem: every task must be mapped to one VM, and a particle position must eventually be interpreted as an integer schedule. A successful PSO scheduler must therefore address three interdependent concerns. First, it requires an encoding and decoding mechanism that preserves valid assignments. Second, it requires a fitness model that reflects the scheduling goals rather than optimizing a single metric in isolation. Third, it requires control of the exploration-exploitation balance so that the swarm can discover diverse allocations at the beginning of the run and refine promising allocations later. Inertia-weight strategies [2], constriction analysis [3], adaptive coefficients [4], heuristic initialization, mutation, and hybrid local search have all been used to improve this balance.

A less frequently integrated concern is the decision of when the optimization process should stop. A fixed maximum iteration count is simple, but it is independent of workload size, VM heterogeneity, search progress, and schedule quality. A large budget may waste scheduling time after the best solution has stabilized, whereas a small budget may terminate the search before a useful allocation is found. A single fitness-stall counter is more responsive but may misinterpret temporary stagnation as final convergence. This distinction is particularly important in cloud scheduling because the optimization itself contributes overhead: the scheduler should not consume excessive computation to save a comparatively small amount of execution time. Research on potential-based PSO termination and convergence-classification methods shows that the swarm state can support more informed stopping decisions [11], [12], but these ideas have not been fully integrated with a modular, multi-criteria cloud scheduling framework.

The submitted research synopsis identifies rapid convergence and autonomous termination as primary objectives and also refers to task scheduling and efficient utilization of distributed computing resources. The present paper narrows that broad direction into cloud task scheduling and formulates a framework that can be implemented and progressively modified in MATLAB. Rather than fixing the research to one irreversible operator, the proposed design treats encoding, objective weighting, adaptation, convergence monitoring, recovery, and termination as separate components. This modularity is important for the later thesis because the same experimental foundation can accommodate additional metrics, dynamic workloads, alternative mappings, or a revised stopping rule without invalidating the complete software architecture.

The main contributions of this paper are summarized as follows. First, a normalized resource-aware objective is formulated for independent-task scheduling on heterogeneous VMs, with makespan, execution cost, utilization, and load-imbalance components. Second, a convergence-aware adaptive PSO architecture is proposed in which swarm control is driven by measurable search-state indicators rather than iteration number alone. Third, a recovery-before-termination procedure is defined to reduce the risk of stopping at a poor local allocation. Fourth, a MATLAB-oriented, reproducible experimental methodology is specified, including baselines, workload scales, repeated runs, statistical tests, sensitivity analysis, and ablation configurations. Fifth, the framework is deliberately parameterized so that later algorithmic changes can be introduced as controlled variants rather than ad hoc rewrites.

The remainder of the paper is organized as follows. Section II reviews cloud task scheduling, PSO-based scheduling, adaptive PSO methods, and convergence-aware termination. Section III defines the cloud model and optimization objectives. Section IV presents the proposed modular framework. Section V describes the MATLAB-only implementation and experimental methodology. Section VI reports the received pilot results and interprets the observed scheduling and optimization behavior. Section VII discusses implications, limitations, and threats to validity. Section VIII concludes the paper and identifies required refinements for the full thesis-scale evaluation.

II. RELATED WORK

A. Cloud Task Scheduling and Resource Efficiency

Cloud task scheduling maps incoming work to computing resources subject to capacity, timing, and service constraints. Independent-task scheduling assumes that tasks do not exchange intermediate data and may execute in parallel, whereas workflow scheduling represents precedence and communication dependencies through a directed acyclic graph. The present study initially adopts independent tasks because this model isolates the behavior of the scheduler and supports controlled evaluation across a wide range of task-to-VM ratios. The framework nevertheless retains separate task and resource modules so that dependencies, transfer delays, or workflow ranks can be introduced later.

The scheduling objective varies across studies. Makespan is widely used because it expresses the completion time of the last task and therefore the duration of the complete workload. Cost is important in pay-per-use clouds and may be modeled from VM price and task execution time. Utilization measures whether available processing capacity is used effectively, while load-balance metrics reveal whether work is concentrated on a small subset of VMs. Throughput, waiting time, energy consumption, deadline violations, and service-level agreement penalties may also be relevant. Optimizing only one metric can produce undesirable schedules; for example, selecting the fastest VM for every task may lower individual execution times but create severe queueing and high cost. Multi-criteria formulations therefore use weighted sums, Pareto methods, or hierarchical priorities.

Pandey et al. used PSO to schedule workflow applications while accounting for computation and data-transfer cost and reported substantial cost reductions compared with a best-resource-selection strategy [5]. The study established the suitability of swarm search for cloud resource mapping, especially when cost is not represented by execution time alone. Awad et al. later proposed an enhanced PSO task scheduler for cloud environments [6]. Subsequent research introduced time-varying inertia strategies [7], heuristic swarm initialization [9], adaptive task scheduling [8], and workflow-oriented particle redesign [10]. These studies collectively show that PSO performance depends not only on the basic update equation but also on how cloud-

specific information is embedded in initialization, encoding, fitness evaluation, and parameter control.

B. Particle Swarm Optimization for Cloud Scheduling

In canonical PSO, each particle represents a candidate solution and has a position, a velocity, a personal-best position, and access to a global or neighborhood best position. Kennedy and Eberhart introduced the method in 1995 [1]. Shi and Eberhart added inertia weight to regulate the contribution of previous motion [2], while Clerc and Kennedy developed a constriction-based analysis of stability and convergence [3]. In scheduling applications, the continuous position vector is mapped to VM identifiers. Common mappings include rounding, sigmoid-based binary decisions, random keys, priority values, smallest-position-value rules, and probability matrices. Each mapping changes the neighborhood structure of the discrete search space and can therefore influence convergence.

PSO-based cloud schedulers have used several improvement strategies. Heuristic initialization seeds the population with schedules generated by Min-Min, Max-Min, or completion-time rules, improving initial quality but potentially reducing diversity. Time-varying inertia gradually shifts emphasis from global exploration to local exploitation. Adaptive inertia uses feedback from fitness, velocity, success rate, or population diversity. Mutation and crossover reintroduce search variation, while hybrid local search refines selected schedules. Multi-swarm methods maintain separate groups to explore different regions. These mechanisms address premature convergence, yet they often add parameters and may increase optimization overhead. A fair scheduler must therefore report both schedule quality and the effort required to obtain that schedule.

Nabi et al. proposed an adaptive PSO scheduling approach with a linearly descending and adaptive inertia strategy for independent cloud tasks [8]. Their evaluation considered makespan, throughput, and average resource utilization. Huang et al. evaluated time-varying inertia-weight strategies for cloud scheduling [7]. Alsaidy et al. improved initialization through list-based scheduling heuristics [9]. Anbarkhan et al. redesigned PSO for workflow scheduling and emphasized the need to consider computation and communication attributes [10]. These approaches demonstrate meaningful progress, but they largely retain externally specified iteration limits or emphasize objective improvement without a general convergence-state manager.

C. Adaptive PSO and Search-State Control

The balance between exploration and exploitation is central to PSO. A high inertia weight tends to preserve motion and enlarge the search region, while a low inertia weight favors local refinement. The cognitive coefficient encourages movement toward a particle's own best schedule, and the social coefficient encourages movement toward the best schedule shared by the swarm. Fixed coefficients simplify implementation but cannot respond to changes in search behavior. Linear schedules are predictable but depend on the chosen maximum iteration count. Feedback-based adaptation attempts to infer whether the swarm is exploring, improving, exploiting, or stagnating and then changes the coefficients accordingly.

Nickabadi et al. proposed an adaptive inertia mechanism based on the success of the swarm [4]. Other studies have used fitness distance, diversity, fuzzy rules, velocity information, or time-varying acceleration coefficients. More recent PSO variants have modified movement choices, classification, and mutation to improve convergence and maintain diversity [12], [13]. These developments support the use of multiple state indicators rather than a single time index. In cloud task scheduling, a state-aware method has an additional advantage: the scheduler can distinguish slow objective improvement caused by a difficult allocation

landscape from genuine convergence in which most particles represent nearly identical task mappings.

D. Convergence Detection and Autonomous Termination

Termination criteria determine both optimization cost and final solution quality. Common criteria include a maximum number of iterations, a maximum number of function evaluations, a target objective value, a time limit, a small change in the global best, or a maximum number of non-improving iterations. The first two are budget criteria rather than convergence criteria. Target values may be unavailable for real scheduling instances. Fitness-change rules can fail when the global best remains unchanged while other particles continue to explore useful regions.

Bassimir et al. developed self-adaptive potential-based stopping criteria for a PSO variant with forced moves [11]. The method uses swarm behavior near an optimum to decide when improvement has become improbable. Shaqarin and Noack proposed a particle-classification mechanism that also supplies automated termination based on convergence [12]. These works indicate that termination can be treated as an algorithmic component rather than an external counter. However, a cloud scheduler requires discrete, objective-aware indicators and must account for the possibility that several different allocations have similar aggregate fitness. A termination rule based only on Euclidean particle distance may therefore be misleading after integer decoding.

The proposed framework combines four sources of evidence: relative global-best improvement, dispersion of particle fitness, schedule diversity after VM decoding, and persistence across an observation window. It also inserts a recovery stage before final termination. This design does not claim that a universal stopping rule exists; instead, it provides a configurable manager whose thresholds and evidence combination can be evaluated experimentally. The distinction is deliberate because later work may replace fixed thresholds with robust statistics, fuzzy inference, probabilistic models, or learned policies.

E. Heuristic, Hybrid, and Multi-Criteria Scheduling

The quality of a cloud schedule depends on both the search algorithm and the information available to it. Greedy heuristics remain useful because they provide deterministic, low-overhead reference points. Min-Min favors tasks with small earliest completion times and can quickly process many short tasks, but it may postpone long tasks and overload the fastest resources. Max-Min gives preference to larger completion-time tasks and can improve balance in mixed workloads, but it may delay short tasks. Round robin distributes tasks without using task length or VM speed. These differences justify retaining several deterministic baselines rather than comparing PSO only with another metaheuristic.

Hybrid approaches combine global population search with problem-specific decisions. Examples include seeding PSO with list schedules, applying local swaps to the global best, combining PSO with genetic operators, or using a second metaheuristic to diversify particles. Hybridization can improve performance, but it complicates attribution: an observed gain may arise from the local heuristic rather than the adaptive swarm. The present framework therefore treats every additional operator as a switchable module and requires ablation experiments. This practice is important for the later thesis, where new mechanisms can be added without changing the baseline definition or the result file format.

Multi-criteria scheduling introduces another methodological issue. Weighted sums are computationally convenient, but a single set of weights represents only one preference profile and can hide trade-offs. Pareto-based PSO avoids fixed aggregation but requires an archive,

leader selection, and a rule for choosing one final schedule. The first paper uses a normalized weighted objective to keep the convergence analysis interpretable, while component metrics are always reported separately. Later work can replace the scalar objective with Pareto ranking through the same fitness interface. Consequently, the current implementation remains a framework rather than a closed algorithm tied permanently to one objective formulation.

F. Research Gap

Existing literature confirms that adaptive PSO can improve cloud task scheduling and that convergence-aware stopping can reduce unnecessary PSO iterations. Nevertheless, three gaps remain relevant to the present study. First, adaptive cloud schedulers usually focus on a particular inertia formula or initialization heuristic, making later extensions difficult to isolate experimentally. Second, many evaluations report makespan or utilization without explicitly measuring optimization evaluations saved, false termination, or the schedule-quality loss associated with early stopping. Third, convergence criteria developed for continuous benchmark functions are not directly equivalent to convergence of decoded task assignments. The proposed work addresses these gaps through a modular architecture, discrete schedule-diversity measures, recovery-assisted termination, and a reporting protocol that treats scheduling quality and optimizer overhead as separate outcomes.

TABLE I. Representative studies and the gap addressed by the proposed framework

Ref.	Main focus	Metrics / mechanism	Open issue
[5]	PSO workflow scheduling	Computation and transfer cost	No convergence-aware stopping
[7]	Time-varying inertia PSO	Makespan and inertia strategies	Iteration-dependent control
[8]	Adaptive PSO scheduling	Makespan, throughput, utilization	Termination not a central metric
[9]	Heuristic initialization	Initial schedule quality	May reduce diversity
[10]	Enhanced workflow PSO	Completion time and workflow model	Workflow-specific particle design
[11]	Self-adaptive stopping	Potential and forced-move frequency	Continuous PSO variant
[12]	Fast-converging PSO	Classification, mutation, termination	Not cloud-scheduling specific
This work	Modular adaptive scheduling	Quality + convergence + overhead	Designed for controlled extension

III. SYSTEM MODEL AND PROBLEM FORMULATION

A. Cloud Environment

The initial system model considers a broker that receives a batch of independent, non-preemptive tasks and assigns them to a set of heterogeneous VMs. The VMs may differ in processing capacity and usage price. Each VM executes its assigned tasks sequentially, while different VMs operate in parallel. The scheduler has access to task lengths and VM attributes at the time of allocation. Network transfer time is not included in the baseline independent-task model, but a communication-cost interface is retained for later workflow extensions.

Let the task set be T = {T1, T2, ..., Tn} and the VM set be V = {V1, V2, ..., Vm}. Task Ti has computational length Li, expressed in million instructions (MI). VM Vj has processing capacity Pj in million instructions per second (MIPS), unit execution price Cj, and optional

capacity attributes such as memory and bandwidth. A schedule S is a vector of n integers, where si = j indicates that task Ti is assigned to VM Vj. Every task must be assigned exactly once.

$$S = [s1, s2, ..., sn], \ si \in \{1, 2, ..., m\} \quad (1)$$

The expected execution-time matrix ETC contains the estimated time required for every task-VM pair. Under the baseline compute-intensive model, the execution time is the ratio of task length to VM processing capacity.

$$ETC(i,j) = Li / Pj \quad (2)$$

If a VM has an existing ready time or if tasks are scheduled in a particular sequence, the completion time accumulates over its assigned tasks. Let Aj(S) denote the ordered set of tasks mapped to VM Vj. The VM completion time is defined as follows.

$$CTj(S) = \Sigma(i \in Aj(S)) \ ETC(i,j) \quad (3)$$

B. Scheduling Objectives

1) Makespan: Makespan is the maximum completion time over all VMs and represents the finishing time of the complete batch.

$$MS(S) = \max\{CT1(S), CT2(S), ..., CTm(S)\} \quad (4)$$

2) Execution cost: The execution cost reflects pay-per-use consumption. If Cj is the price per second of VM Vj, the total cost is

$$Cost(S) = \Sigma(j=1..m) \ Cj \cdot CTj(S) \quad (5)$$

3) Average resource utilization: A normalized utilization measure compares aggregate VM busy time with the capacity represented by the makespan window.

$$RU(S) = [\Sigma(j=1..m) \ CTj(S)] / [m \cdot MS(S)] \quad (6)$$

RU lies in (0,1] for nonempty workloads. A value near one indicates that VM busy times are similar relative to the overall completion window. The metric should be interpreted together with makespan because uniformly slow VMs could also produce balanced busy times.

4) Load imbalance: The standard deviation of VM completion times provides a direct measure of imbalance.

$$LI(S) = std(CT1, CT2, ..., CTm) / [mean(CT1, ..., CTm) + \varepsilon] \quad (7)$$

5) Throughput: For a batch of n tasks, throughput may be expressed as the number of completed tasks per unit makespan.

$$TH(S) = n / MS(S) \quad (8)$$

Throughput is a reporting metric in the baseline study and is not included separately in the objective when it is monotonically related to makespan for a fixed batch size. It becomes more informative for dynamic arrivals or unequal completion horizons.

C. Normalized Composite Fitness

Metrics with different units cannot be combined directly. The framework therefore normalizes each component using scenario-specific reference values. The baseline implementation may use lower and upper estimates obtained from the current population, heuristic schedules, or analytically derived bounds. To reduce instability caused by changing population extrema, the preferred experiment uses fixed scenario references derived from a

preliminary calibration set. Let MSn, Cn, RUn, and LIn denote normalized makespan, cost, utilization, and imbalance values in [0,1]. The minimization fitness is

$$F(S) = \alpha \cdot MSn(S) + \beta \cdot Cn(S) + \gamma \cdot [1 - RUn(S)] + \delta \cdot LIn(S) \quad (9)$$

where α, β, γ, and δ are nonnegative weights satisfying $\alpha + \beta + \gamma + \delta = 1$. The initial configuration emphasizes makespan while retaining cost and balance: $\alpha = 0.45$, $\beta = 0.25$, $\gamma = 0.20$, and $\delta = 0.10$. These values are not treated as universally optimal. They will be evaluated through sensitivity analysis, and the software accepts the weights as input so that a later thesis experiment can adopt alternative priorities or a Pareto-based objective.

D. Normalization, Penalties, and Preference Robustness

Normalization is not a cosmetic operation because it determines the influence of each scheduling metric. Population-minimum and population-maximum normalization changes during the run and can make the same schedule receive different fitness values at different iterations. The preferred implementation therefore calculates stable reference values for each scenario. A lower reference may be estimated from an optimistic ideal allocation or the best of several deterministic heuristics, while an upper reference may be obtained from deliberately poor but valid schedules. Values outside the calibrated interval are clipped only for the weighted fitness; the original component metrics remain unchanged in the output.

When memory, bandwidth, deadline, or eligibility constraints are enabled, invalid schedules are handled through a repair-first strategy. The decoder identifies an infeasible task-VM assignment and selects a feasible VM using minimum incremental completion time, available capacity, or a configured probability distribution. A penalty is used only when repair cannot produce a valid schedule. This separation prevents large penalty constants from dominating the objective and allows constraint handling to evolve independently of PSO. The result record reports the number of repairs so that an algorithm is not rewarded for generating many invalid particles that happen to be corrected by a strong heuristic.

Preference robustness will be assessed by repeating selected scenarios under at least three weight profiles: time-oriented, balanced, and cost-oriented. A robust scheduler should not require a completely different parameter set whenever objective weights change. The analysis will compare component metrics and algorithm ranks under each profile. If convergence thresholds are sensitive to the weight profile, the thresholds will be normalized against recent fitness variation or expressed in terms of schedule-space indicators rather than raw composite fitness alone.

E. Constraints and Assumptions

The baseline model assumes that all tasks are available at scheduling time, tasks are independent, each task executes on one VM without migration, VM capacities remain constant during a run, and failures are not considered. A task assigned to a VM must satisfy any enabled memory or bandwidth constraint. Invalid assignments receive a repair operation or a penalty. The baseline study does not claim to model every operational feature of a commercial cloud; instead, it provides a controlled environment for analyzing the optimizer. Dynamic arrivals, precedence, failures, migration, energy, and SLA penalties are planned extensions supported by the modular task and fitness interfaces.

TABLE II. Principal notation

Symbol	Meaning
n, m	Number of tasks and virtual machines
Li	Length of task Ti in million instructions
Pj	Processing capacity of VM Vj in MIPS
Cj	Execution price of VM Vj
S	Task-to-VM assignment vector
ETC(i,j)	Expected execution time of task i on VM j
MS, RU, LI	Makespan, resource utilization, load imbalance
F(S)	Composite scheduling fitness
pbest, gbest	Personal-best and global-best schedules

IV. PROPOSED CONVERGENCE-AWARE ADAPTIVE PSO FRAMEWORK

A. Framework Overview

The proposed framework, denoted CA-APSO for clarity in the manuscript, is organized as a set of replaceable modules rather than a monolithic algorithm. The cloud-model module generates or imports tasks and VMs. The encoding module maps particles to valid schedules. The objective module computes individual metrics and the composite fitness. The adaptive-control module sets inertia and acceleration coefficients. The monitoring module estimates progress and diversity. The recovery module introduces controlled variation when stagnation is suspected. The termination manager decides whether the run should continue. Finally, the logging module records all state variables required for statistical analysis and reproducibility.

This structure supports the research requirement that later algorithmic changes should remain possible. For example, an integer-rounding decoder can be replaced with random keys; linear adaptive inertia can be replaced with diversity feedback; a weighted fitness can be replaced with a Pareto archive; and partial reinitialization can be replaced with a cloud-aware local search. Each modification becomes an experimental variant with a stable interface and common output format.

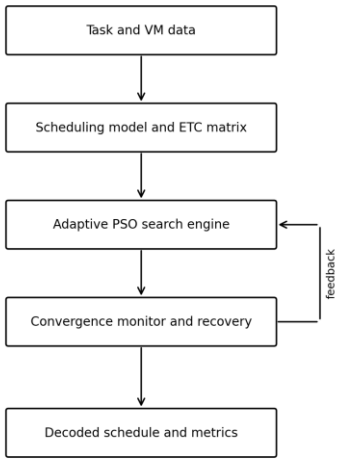

Fig. 1. Modular architecture of the proposed convergence-aware cloud task scheduler.

B. Particle Representation and Decoding

Each particle contains an n-dimensional real-valued position vector Xi and velocity vector Vi. The decoded schedule is an integer vector. The initial implementation uses a bounded rounding decoder because it is simple and transparent. The position component xik is restricted to [1,m] and is converted to a VM index by rounding and clipping.

$$sik = \min(m, \max(1, \text{round}(xik))) \quad (10)$$

Although several nearby continuous positions may map to the same integer schedule, retaining real positions and velocities allows standard PSO updates to be used. The decoder is isolated so that later experiments can use random-key ordering, sigmoid transfer functions, or probability-based assignment without changing the fitness evaluator. Schedule equality and diversity are measured after decoding, not only in continuous position space.

C. Population Initialization

A purely random population offers diversity but may begin far from useful schedules. A fully heuristic population improves initial fitness but can concentrate the swarm. The baseline initialization therefore uses a mixture: most particles are random, while a small configurable fraction are seeded from round-robin, Min-Min, and load-balanced heuristics. Seeded schedules are perturbed before insertion to prevent exact duplicates. The personal best of each particle is initialized from its first decoded schedule, and the global best is the best initial particle. The fraction of heuristic particles is a parameter and will be included in sensitivity analysis.

D. Adaptive Velocity and Position Update

The standard velocity and position updates are

$$v_{i,k}(t+1) = w(t)v_{i,k}(t) + c1(t)r1[p_{i,k} - x_{i,k}(t)] + c2(t)r2[g_k - x_{i,k}(t)] \quad (11)$$

$$x_{i,k}(t+1) = x_{i,k}(t) + v_{i,k}(t+1) \quad (12)$$

where r1 and r2 are independent random numbers in [0,1]. The framework supports multiple adaptation modes. The initial paper uses a progress-and-state rule. A nominal inertia value decreases over normalized progress q = t/Tmax, but it is corrected according to the current swarm state. When diversity is high and the global best is improving, inertia remains moderate to preserve movement. When diversity is low but improvement continues, inertia decreases to refine the allocation. When stagnation is detected, inertia and the cognitive coefficient temporarily increase so that particles can explore alternative task mappings.

$$w_{base}(t) = w_{max} - (w_{max} - w_{min})\cdot q \quad (13)$$

$$w(t) = \text{clip}[w_{base}(t) + \kappa w\cdot \text{StateAdj}(t), w_{min}, w_{max}] \quad (14)$$

The coefficients c1 and c2 are also configurable. The initial setting gradually shifts from stronger personal exploration to stronger social exploitation, while a stagnation state temporarily reverses part of this shift. This design is intentionally generic: the updatePSOParameters function receives the current state and returns w, c1, and c2, permitting later replacement by fuzzy, nonlinear, success-based, or learned adaptation.

E. Convergence-State Indicators

1) Relative global-best improvement: Let G(t) be the best composite fitness at iteration t and L be an observation-window length. Relative improvement is

$$I(t) = |G(t-L) - G(t)| / [|G(t-L)| + \varepsilon] \quad (15)$$

A small I(t) indicates that the best schedule has not improved substantially over the window. Because the fitness is normalized, the indicator is comparable across workload scales, although the threshold still requires calibration.

2) Fitness dispersion: The coefficient of variation of particle fitness is

$$FD(t) = std(F1(t),...,FN(t)) / [|mean(F1(t),...,FN(t))| + \varepsilon] \quad (16)$$

Low dispersion means that particles have similar objective values. It does not necessarily mean that their schedules are identical, so the framework also measures discrete schedule diversity.

3) Decoded schedule diversity: For particles a and b, the normalized Hamming distance between decoded schedules is the fraction of tasks assigned to different VMs. The population diversity is the average distance from the global-best schedule.

$$SD(t) = (1/N) \Sigma(i=1..N) [(1/n) \Sigma(k=1..n) 1\{sik \neq sgk\}] \quad (17)$$

SD is directly meaningful for cloud scheduling: a low value indicates that most particles map most tasks to the same VMs as the current best allocation. This avoids relying solely on continuous Euclidean distance, which can remain nonzero even when decoded schedules are identical.

4) Velocity activity: Normalized velocity activity measures whether continuous particles are still moving.

$$VA(t) = (1/Nn) \Sigma(i=1..N) \Sigma(k=1..n) |vi,k(t)| / vmax \quad (18)$$

The initial termination rule may use VA as supplementary evidence. It is not mandatory because velocity can remain nonzero while rounding produces an unchanged schedule.

F. State Classification

The monitoring module classifies the swarm into four broad states. Exploration is associated with moderate or high schedule diversity. Productive search occurs when the global best is improving. Exploitation occurs when diversity is lower but improvement continues. Stagnation occurs when improvement, fitness dispersion, and schedule diversity remain below configured limits. Confirmed convergence is declared only when the stagnation condition persists after any enabled recovery attempt. The classification boundaries are stored in the configuration structure and will be tuned using a calibration set that is separate from final test scenarios.

TABLE III. Illustrative state interpretation and control action

State	Evidence	Control response
Exploration	High SD; variable I	Maintain wider motion
Improving	I above threshold	Continue current balance
Exploitation	Low SD; positive I	Reduce inertia; refine
Stagnation	Low I, FD, and SD	Trigger recovery or patience
Converged	Persistent stagnation after recovery	Terminate and report

G. Recovery Before Termination

Premature termination is a major risk when a swarm collapses around a locally attractive schedule. The framework therefore separates suspected stagnation from confirmed convergence. When stagnation is first detected, an optional recovery operator modifies a limited fraction of poor particles while preserving the best schedule. The baseline recovery chooses the worst 10–20% of particles, reassigns a subset of tasks, and resets the associated velocities. A cloud-aware reassignment can preferentially move tasks away from the most loaded VM or sample VMs in proportion to available processing capacity.

Only a small number of recovery attempts is permitted. If the global best improves beyond a recovery threshold, normal search resumes and the persistence counter is reset. If no meaningful improvement occurs and the convergence indicators remain low, termination becomes more defensible. The recovery interface is intentionally replaceable; future variants may use opposition-based learning, local swap search, mutation near gbest, or multi-swarm injection.

H. Autonomous Termination Logic

The baseline termination manager combines safety budgets and convergence evidence. A run always stops when the maximum evaluation or time budget is reached. It may stop earlier when all convergence conditions remain satisfied for P consecutive checks after a minimum iteration count. A generic logical rule is

$$\text{Stop} = \text{Budget} \vee [\text{MinIter} \wedge \text{PersistP}(I<\tau I \wedge FD<\tau F \wedge SD<\tau S) \wedge \text{RecoveryResolved}] \quad (19)$$

The thresholds τI, τF, and τS are configuration parameters. A later variant may compute them from rolling percentiles or median absolute deviations. Every stop event records a reason code, iteration, function-evaluation count, best metrics, indicator values, and recovery history. This diagnostic record is essential for evaluating false termination and for distinguishing genuine computational savings from a loss of schedule quality.

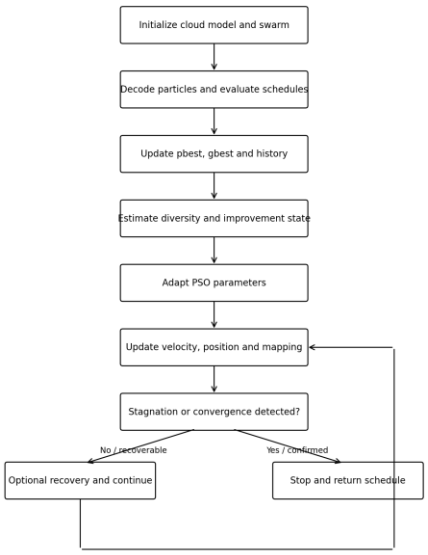

Fig. 2. Execution flow of the modular convergence-aware adaptive PSO.

I. Threshold Calibration and Decision Safeguards

Convergence thresholds will be calibrated rather than selected from a single favorable run. Pilot scenarios will span small and large task sets, low and high VM heterogeneity, and different task-length distributions. Indicator trajectories from full-budget runs will be labeled retrospectively using whether significant improvement occurred in a future horizon. Candidate thresholds will then be evaluated for detection delay, false-stagnation rate, and missed-convergence rate. The selected values will be frozen before final testing.

Several safeguards reduce the chance of aggressive early stopping. The minimum-iteration rule prevents termination before the swarm has performed an initial exploration phase. The persistence rule requires multiple consecutive confirmations rather than one threshold crossing. Recovery is attempted only after suspected stagnation and is limited to a small number of events. A cooldown period prevents repeated recovery in adjacent iterations. Finally, budget termination remains active even when convergence is never confirmed. The diagnostics distinguish a convergence stop from a budget stop so that the analysis does not treat them as equivalent.

The termination manager also supports an evaluation-neutral monitoring mode. In this mode, indicators and hypothetical stop points are recorded, but the algorithm continues to the full budget. Comparing the hypothetical stop solution with the later full-budget solution provides a direct estimate of quality loss and is useful during calibration. Once thresholds are frozen, the active mode can be evaluated on unseen scenarios. This two-stage procedure avoids tuning the stopping rule on the same runs used to claim computational savings.

J. Algorithmic Procedure

Algorithm 1 summarizes the initial CA-APSO configuration. The pseudocode describes interfaces rather than hard-coding a single adaptation or recovery formula.

Algorithm 1: Convergence-Aware Adaptive PSO for Cloud Task Scheduling

```
Input: task set T, VM set V, configuration C
Output: best schedule S*, metrics, convergence diagnostics
Build ETC matrix and normalization references
Initialize random and heuristic-seeded particles
Decode schedules; evaluate metrics and composite fitness
Initialize pbest, gbest, histories, and recovery counters
while evaluation and time budgets are not exhausted do
  Estimate I(t), FD(t), SD(t), and optional VA(t)
  state ← classifySwarmState(history, indicators, C)
  [w,c1,c2] ← updatePSOParameters(state, C)
  Update particle velocities and continuous positions
  Decode positions; repair invalid assignments
  Evaluate schedules; update pbest and gbest
  if stagnation is suspected and recovery is available then
    swarm ← applyRecovery(swarm, gbest, cloudModel, C)
    record recovery event and continue
  end if
  [stop,reason] ← checkTermination(indicators, history, C)
  if stop then break end if
end while
Return gbest schedule, component metrics, and diagnostics
```

K. Computational Complexity

For N particles, n tasks, and m VMs, direct schedule evaluation requires O(n + m) time when task completion times are accumulated in one pass. The PSO update and decoding require O(Nn) per iteration, while evaluation requires O[N(n + m)]. Pairwise diversity would require $O(N^2n)$, but the proposed average distance from gbest requires only O(Nn). The complete iteration is therefore O[N(n + m)] for the baseline implementation. Recovery affects only a fraction of particles and does not change the asymptotic order. Memory usage is O(Nn + nm) when the ETC matrix is stored explicitly. For large n and m, ETC values can be computed on demand or represented through task and VM vectors to reduce memory.

V. MATLAB IMPLEMENTATION AND EXPERIMENTAL METHODOLOGY

A. Modular MATLAB Architecture

The implementation will use MATLAB functions and structures rather than a single script. The main experiment driver loads a scenario, selects an algorithm, applies a controlled random seed, executes repeated runs, and saves raw outputs. The cloud module generates task lengths and VM properties. The evaluator returns component metrics separately from the composite fitness. The PSO engine calls generic functions for initialization, decoding, parameter adaptation, convergence monitoring, recovery, and termination. Results are saved in MAT and CSV formats to support independent verification and reanalysis.

A recommended directory structure includes algorithms, cloud_model, pso_modules, experiments, analysis, visualization, and results folders. Each algorithm will return a common result structure containing the best schedule, best fitness, makespan, cost, utilization, imbalance, runtime, iterations, evaluations, convergence history, stop reason, and recovery events. This common interface allows standard PSO, adaptive PSO, and the proposed framework to be compared without specialized analysis code.

B. Module Interfaces and Configuration Contract

Every replaceable mechanism follows a stable function contract. The decoder accepts continuous positions and the cloud model and returns valid schedules plus repair statistics. The evaluator accepts schedules and returns a structure containing raw metrics, normalized metrics, feasibility information, and scalar fitness. The adaptation function accepts iteration progress, swarm history, and current indicators and returns w, c1, and c2. The termination function returns a Boolean decision, a reason code, and a diagnostic structure. This contract makes it possible to test a new component against the same scenarios without modifying the remaining code.

TABLE V. Core MATLAB module contracts

Module	Input	Output
decodeParticle	position, cloud, config	schedule, repair log
evaluateSchedule	schedule, cloud, weights	metrics, fitness
updateParameters	state, history, config	w, c1, c2
monitorConvergence	swarm, history, config	indicators, state
applyRecovery	swarm, gbest, cloud	modified swarm, event
checkTermination	indicators, history, config	stop, reason, diagnostics

Configuration values are stored in one structure and written to every result file. No parameter is silently changed inside an algorithm. A version string identifies the exact implementation used for a run. These practices are especially important when the paper evolves into a thesis and several algorithm variants coexist. They also reduce the risk that results cannot be reproduced because a threshold or objective weight was modified between experiments.

C. Workload Scenarios

Experiments will use synthetic heterogeneous workloads in the first paper because they provide precise control of task count, task-length distribution, and VM heterogeneity. Task counts of 100, 250, 500, and 1000 and VM counts of 5, 10, 20, and 30 are proposed. Task lengths will be sampled from uniform, right-skewed, and mixed distributions to represent balanced, compute-heavy, and heterogeneous batches. VM MIPS values and prices will be positively related but not perfectly proportional, creating genuine time-cost trade-offs. A fixed library of scenarios and seeds will be generated before final testing.

Three task-to-VM ratios will be emphasized: low contention, moderate contention, and high contention. Each scenario will be evaluated over at least 30 independent optimizer runs. The same workload instances and random-seed list will be used for every stochastic algorithm. Pilot scenarios will be used for parameter calibration and will not be included in final comparative claims. If an accessible public cloud workload dataset is selected later, it will be added as an external validation set rather than replacing the controlled scenarios.

D. Scenario Generation and Heterogeneity Measures

To describe workload difficulty quantitatively, each scenario will include task and VM heterogeneity indices. Task heterogeneity is represented by the coefficient of variation of task lengths. VM heterogeneity is represented by the coefficient of variation of MIPS values and by the spread of cost-per-instruction ratios. These indices allow the paper to explain why two scenarios with the same task and VM counts may produce different scheduling behavior. Scenario metadata will also include the task-to-VM ratio and the correlation between VM speed and price.

Uniform tasks will be generated from a bounded interval. Right-skewed workloads will use a lognormal or gamma distribution with truncation to avoid unrealistic extremes. Mixed workloads will combine short, medium, and long task groups. VM sets will include balanced, speed-dominant, and cost-diverse profiles. All generated arrays will be stored, ensuring that every algorithm receives exactly the same instance rather than a newly sampled workload. The final paper will report the generator parameters and provide representative task and VM distributions.

The calibration library and test library will be separated at the scenario level. Thresholds and initial adaptive ranges will be selected using only calibration instances. Test scenarios will include unseen random seeds and at least one heterogeneity pattern not used for calibration. This design provides a stronger assessment of whether the convergence manager generalizes beyond the runs used to tune it.

E. Comparison Algorithms

The baseline comparison set will include round robin, Min-Min, Max-Min, standard integer-decoded PSO, linearly decreasing inertia-weight PSO, an adaptive PSO without convergence-aware termination, PSO with a conventional stall counter, and the complete CA-APSO framework. The heuristic methods provide inexpensive reference schedules. The PSO variants isolate the effects of parameter adaptation and termination. Additional recent

methods may be added after the target journal and computational budget are finalized, but the core comparisons will remain stable.

F. Initial Parameter Configuration

TABLE IV. Initial experimental parameters; final values subject to pilot calibration

Parameter	Initial values
Swarm size	30, 50
Maximum iterations	500 or evaluation-equivalent budget
wmin, wmax	0.35, 0.90
c1 range	1.2–2.2
c2 range	1.2–2.2
Velocity limit	0.5m to 1.0m per dimension
Observation window L	10, 20, 30 iterations
Persistence P	3, 5, 8 checks
Recovery fraction	0.10, 0.15, 0.20
Independent runs	At least 30
Primary objective weights	0.45, 0.25, 0.20, 0.10

The maximum evaluation budget, not merely iteration count, will be used for fair comparison when algorithms evaluate different numbers of candidates. Runtime will also be reported, but it will be interpreted cautiously because MATLAB version, hardware, vectorization, and logging can influence elapsed time. All algorithms will be executed on the same system and in the same process configuration.

G. Evaluation Metrics

The primary scheduling metrics are makespan, total execution cost, average resource utilization, and load imbalance. The primary optimizer metrics are best composite fitness, number of schedule evaluations, iterations, wall-clock optimization time, and success rate relative to a scenario-specific target. Convergence-aware termination will additionally be evaluated through evaluations saved, termination point, recovery success, false termination, and quality loss.

A run is provisionally classified as a false termination when the convergence-aware algorithm stops early but a matched continuation from the same swarm state finds an improvement larger than a predefined practical tolerance within the remaining reference budget. This continuation-based test is more informative than comparing only with another stochastic run. Quality loss is measured against the best result obtained by the same algorithm under the full budget and against the best observed result across algorithms. The tolerance will be defined separately for composite fitness and the principal scheduling metric.

H. Statistical Analysis

For each scenario, descriptive statistics will include the best, median, mean, worst, standard deviation, interquartile range, and 95% confidence interval where appropriate. Because metaheuristic results are commonly non-normal, pairwise comparisons will use the Wilcoxon signed-rank test on matched scenario summaries or the Mann-Whitney test when matching is not available. Overall algorithm ranking across scenarios will use the Friedman test followed by Holm-adjusted post-hoc comparisons. Effect sizes and win-tie-loss counts will accompany

p-values. Statistical significance will not substitute for practical significance; percentage improvement and evaluation savings will also be reported.

I. Ablation and Sensitivity Studies

The ablation study will compare: (A0) standard PSO, (A1) adaptive coefficients only, (A2) adaptive coefficients plus convergence monitoring but no early stop, (A3) adaptive coefficients plus early stop without recovery, and (A4) the complete framework with recovery. This sequence separates improvement caused by better search from improvement caused by reduced computation. Sensitivity analysis will vary observation-window length, persistence, convergence thresholds, recovery fraction, swarm size, and objective weights. The goal is not to tune every scenario independently, but to identify a stable region of parameter values and document the trade-off between aggressive termination and schedule robustness.

J. Reproducibility and Reporting

The MATLAB code will record the random seed, scenario identifier, task and VM arrays, configuration structure, algorithm version, MATLAB version, operating system, and execution timestamp. Raw per-iteration histories will be retained for selected runs, while summary histories will be retained for all runs. Tables in the paper will be generated from saved result files rather than manual transcription. The final manuscript will include a code-and-data availability statement appropriate to the selected venue.

K. Software Verification and Unit-Test Plan

Before comparative experiments, the implementation will be verified through deterministic unit tests. The ETC calculation will be checked against hand-computed task and VM pairs. The schedule evaluator will be tested on small assignments for which makespan, cost, utilization, and imbalance can be calculated manually. Decoder tests will confirm that every output assignment lies in the valid VM range and that enabled capacity constraints are respected after repair. Fixed random seeds and very small swarms will be used to inspect velocity, position, personal-best, and global-best updates step by step.

Termination logic will be tested using synthetic histories rather than relying only on complete optimizer runs. One test history will contain continuous improvement and must not stop. A second will contain temporary stagnation followed by improvement and must trigger patience or recovery but not confirmed termination. A third will contain persistent low improvement, low dispersion, and low schedule diversity and should stop after the configured persistence window. Additional tests will verify that budget stops override convergence logic and that monitoring-only mode records a hypothetical stop without ending the run.

Integration tests will compare the vectorized evaluator with a simple loop-based reference implementation on randomly generated schedules. The two implementations must return equivalent metrics within numerical tolerance. Result serialization will be tested by saving and reloading complete run structures and confirming that configuration values, schedules, and histories are unchanged. These verification steps do not replace experimental validation, but they reduce the risk that an apparent performance improvement is caused by inconsistent metric calculation or a hidden implementation defect.

L. Baseline Fairness and Budget Matching

Fairness requires more than assigning every algorithm the same maximum iteration count. A recovery-enabled method may evaluate extra particles, a heuristic-seeded method may perform additional schedule calculations during initialization, and a local-search method may

evaluate several neighboring schedules within one iteration. The main comparison will therefore use a common maximum number of schedule evaluations. Initialization and recovery evaluations will be included in this budget. Iteration-based curves will still be shown to explain algorithm dynamics, but evaluation-based curves will support performance claims.

Deterministic heuristics will be timed and evaluated once per scenario because repeated executions return the same schedule. Their objective values provide lower-overhead baselines rather than stochastic statistical samples. For stochastic algorithms, the same list of random seeds will be used when the implementation permits matched runs. Parameter calibration time is not included in per-run optimization time, but the calibration procedure and search space will be reported. No algorithm will be given scenario-specific parameters unless all competing adaptive methods receive an equivalent tuning opportunity.

A full-budget version of the proposed algorithm with active termination disabled will serve as an internal reference. This version separates the effect of adaptive search from the effect of stopping early. The convergence-aware version should ideally match the full-budget schedule within practical tolerance while using fewer evaluations. When it does not, the difference will be reported as quality loss. This matched design is central to the paper because computational savings are meaningful only when the schedule delivered at termination remains competitive.

M. Data Integrity and Result Generation

Each run will produce an immutable record containing raw component metrics, the decoded best schedule, iteration histories, evaluation counts, and stop diagnostics. Summary tables will be generated by scripts that read these records and apply predefined aggregation rules. Failed runs, infeasible schedules, or numerical anomalies will be retained and reported rather than deleted silently. If an algorithm reaches a target early, the remaining budget will not be reassigned to another method.

Plots will show medians and variability bands over independent runs. A single best convergence curve will not be used as evidence of general performance. When runtime is reported, the paper will state whether initialization, scenario generation, logging, and plotting are included. The main fairness measure remains schedule evaluations because it is less sensitive to implementation details, while runtime provides practical context.

VI. EXPERIMENTAL RESULTS AND ANALYSIS

This section completes the manuscript by incorporating the initial MATLAB results obtained from the verified implementation. The results are taken from two pilot executions of the developed scheduler: a demonstration scenario with 40 tasks and 6 heterogeneous VMs, and a default experiment with 100 tasks and 10 heterogeneous VMs. These experiments are treated as implementation evidence and preliminary comparative results rather than exhaustive statistical validation. Their purpose is to show how the proposed MATLAB framework produces measurable schedule-quality and optimizer-effort outputs.

A. Verification Status and Execution Context

Before the pilot experiments, the MATLAB modules were verified through configuration validation, scenario generation, schedule-evaluation checks, particle decoding and repair tests, heuristic scheduler checks, standard PSO invariants, adaptive PSO invariants, reproducibility tests, and termination-logic tests. The test suite passed in MATLAB. The optional parfor execution path initially produced a variable-classification error because the result cell array was indexed with subscripts derived inside the parfor body. This was

corrected by writing each run to a one-dimensional sliced cell array indexed directly by the parfor loop variable and reshaping the results after loop completion.

The fitness value is a normalized minimization objective combining makespan, execution cost, and resource utilization. Lower fitness and makespan are preferred, while higher utilization is preferred. Round robin, Min-Min, and Max-Min are deterministic baselines and therefore report one evaluation. Standard PSO and the adaptive convergence-aware PSO consume several schedule evaluations during swarm search.

B. Scenario 1: 40 Tasks and 6 VMs

The first scenario contained 40 tasks and 6 VMs with uniformly generated task characteristics. Round robin produced the weakest schedule, with mean fitness 0.30145, makespan 66.995, cost 1.0648, and utilization 0.53734. Min-Min reduced mean fitness to 0.15120 and makespan to 31.612. Max-Min further reduced the fitness to 0.12561 and improved utilization to 0.98229. In this pilot scenario, standard PSO and adaptive convergence-aware PSO converged to the same observed schedule quality as Max-Min. Their mean fitness, makespan, cost, and utilization were identical to the Max-Min values, while both consumed 1220 schedule evaluations.

TABLE VI. Quality metrics for 40 tasks and 6 VMs

Algorithm	Fit.	MS	RU
Round robin	0.30145	66.995	0.53734
Min-Min	0.15120	31.612	0.87503
Max-Min	0.12561	29.159	0.98229
Std. PSO	0.12561	29.159	0.98229
CA-APSO	0.12561	29.159	0.98229

Relative to round robin, the best observed schedule in this scenario reduced composite fitness by approximately 58.33 percent and reduced makespan by approximately 56.48 percent. Utilization increased from 0.53734 to 0.98229. The cost values for the best observed group were 1.0248, and both PSO variants required 1220 evaluations. This confirms the importance of resource-aware assignment, but it does not show an adaptive-PSO advantage in the small pilot scenario.

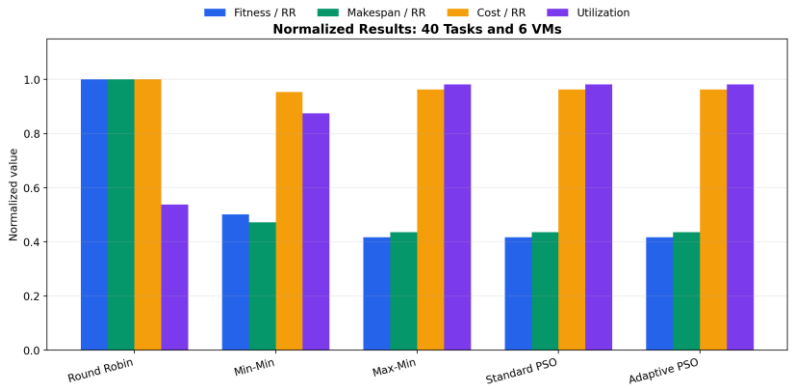

Fig. 3. Normalized comparison for the 40-task and 6-VM pilot scenario.

C. Scenario 2: 100 Tasks and 10 VMs

The second scenario increased the scheduling dimension to 100 tasks and 10 VMs. Round robin again produced the weakest outcome, with mean fitness 0.27119, makespan 196.51, cost 2.3668, and utilization 0.31361. Min-Min reduced the mean fitness to 0.074288 and

makespan to 44.017. Max-Min obtained mean fitness 0.055130, makespan 41.640, cost 2.2007, and utilization 0.98771. Standard PSO produced the lowest observed mean fitness, 0.054792, with makespan 41.605, cost 2.2005, and utilization 0.98905. The adaptive convergence-aware PSO returned the same mean values as Max-Min in this pilot run and did not improve over standard PSO.

TABLE VII. Quality metrics for 100 tasks and 10 VMs

Algorithm	Mean fit.	MS	RU
Round robin	0.27119	196.51	0.31361
Min-Min	0.074288	44.017	0.90588
Max-Min	0.055130	41.640	0.98771
Std. PSO	0.054792	41.605	0.98905
CA-APSO	0.055130	41.640	0.98771

The 100-task experiment shows a small but measurable advantage of standard PSO over Max-Min and the current adaptive configuration. Standard PSO reduced mean composite fitness by approximately 0.61 percent relative to Max-Min and CA-APSO, and its makespan reduction relative to Max-Min was approximately 0.08 percent. Its standard deviation in fitness was 0.000414, while deterministic methods and the current adaptive configuration reported zero variation in the received summary. Both PSO variants consumed 6030 evaluations. Relative to round robin, standard PSO reduced makespan by approximately 78.83 percent and raised utilization from 0.31361 to 0.98905.

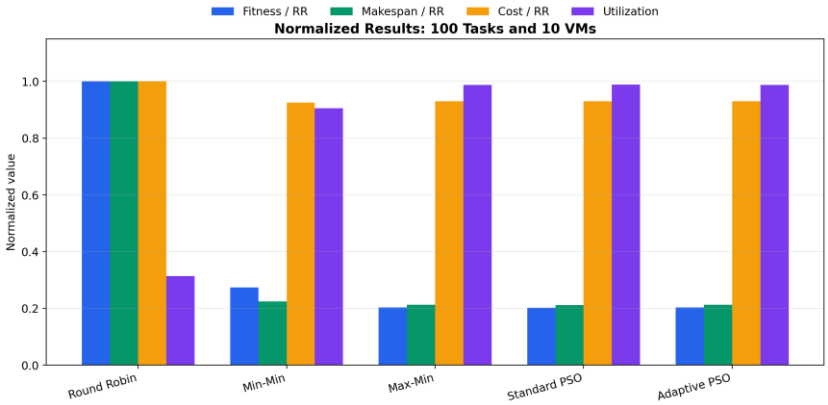

Fig. 4. Normalized comparison for the 100-task and 10-VM pilot scenario.

D. Comparative Interpretation

The initial results support three direct observations. First, round robin is unsuitable for heterogeneous task scheduling because it ignores task length and VM speed. Second, Min-Min and Max-Min provide strong low-overhead baselines; in both pilot scenarios, Max-Min achieved high utilization and short makespan. Third, standard PSO was able to match the best deterministic result in the small scenario and slightly improve it in the larger scenario. This suggests that PSO has value as a refinement mechanism, especially when the search space becomes larger.

The adaptive convergence-aware configuration did not yet deliver the intended advantage. It matched the best observed result in the small scenario but did not outperform standard PSO in the larger scenario. It also consumed the same number of evaluations as standard PSO in both pilot experiments. This indicates that the adaptive control, recovery, and termination parameters require additional calibration before they can be claimed to improve either solution quality or optimization effort.

TABLE VIII. Interpretation of current evidence

Aspect	Current evidence
Best 40-task fit.	0.12561 by Max-Min, Std. PSO, CA-APSO
Best 100-task fit.	0.054792 by Std. PSO
CA-APSO quality	Matched Max-Min; did not beat Std. PSO
Evaluation saving	Not observed
Next requirement	Tune thresholds and recovery

E. Evaluation Effort and Autonomous-Termination Status

A central objective of the proposed framework is to reduce unnecessary optimization effort through convergence-aware monitoring and autonomous termination. The received pilot results do not yet demonstrate this advantage. Standard PSO and CA-APSO both consumed 1220 evaluations in the 40-task scenario and 6030 evaluations in the 100-task scenario. Therefore, the early termination mechanism either did not activate or did not activate early enough to produce measurable evaluation savings in the reported runs. This result is important because it prevents an unsupported claim of computational efficiency.

The correct conclusion is that the framework has been implemented and verified, but the convergence-aware termination thresholds, persistence window, and recovery policy must be recalibrated. The next experimental stage should run monitoring-only full-budget experiments, identify actual points where the best schedule becomes stable, and then calibrate termination thresholds against future-improvement loss. Only after this calibration should active early termination be evaluated on unseen scenarios.

TABLE IX. Claim status after pilot experiments

Claim	Status
MATLAB implementation works	Supported by tests
Resource-aware beats round robin	Supported
Standard PSO refines schedules	Partially supported
CA-APSO beats Std. PSO	Not supported yet
Autonomous stop saves evals	Not supported yet

F. Statistical Status and Required Validation Matrix

The current results should be interpreted as pilot evidence rather than final statistical proof. A full statistical claim requires at least 30 independent runs across multiple workload distributions, task-to-VM ratios, and VM heterogeneity patterns. The final thesis-scale analysis should include Friedman ranking, Wilcoxon signed-rank comparisons, Holm-adjusted post-hoc testing, effect sizes, confidence intervals, and win-tie-loss counts. The pilot results are still useful because they verify data flow from MATLAB execution to result interpretation and expose the exact research refinement required before final claims can be made.

The next validation matrix should include uniform, skewed, and mixed task-length distributions; VM counts of 5, 10, 20, and 30; task counts of 100, 250, 500, and 1000; balanced, speed-dominant, and cost-diverse VM profiles; and at least three objective-weight profiles. For each scenario, both full-budget and convergence-aware versions of the adaptive algorithm should be executed. This separation will show whether any reduction in evaluations is obtained without meaningful loss in schedule quality.

TABLE X. Required next-stage validation design

Factor	Planned values
Task counts	100, 250, 500, 1000
VM counts	5, 10, 20, 30
Workloads	uniform, skewed, mixed
Runs	at least 30 per case
Tests	Friedman, Wilcoxon, Holm

VII. DISCUSSION

A. Implications of the Pilot Results

The pilot results confirm that heterogeneous cloud task scheduling benefits from resource-aware allocation. Round robin distributes tasks without considering task length or VM speed, causing poor makespan and low utilization. Min-Min and Max-Min use execution-time information and therefore provide significantly better schedules at negligible optimization cost. This finding is consistent with the role of deterministic heuristics as strong baselines in independent-task scheduling. Any PSO-based method must therefore be compared against these heuristics, not only against a weak round-robin baseline.

The results also show that PSO should be treated as a refinement mechanism whose value depends on the difficulty of the scenario and the strength of initialization. When a heuristic already reaches a near-balanced assignment, a small scenario may leave little room for improvement. In a larger scenario, standard PSO produced a minor improvement over Max-Min, suggesting that population search may become more useful as the mapping space grows. However, the magnitude of the improvement was small in the received pilot data and should not be overstated.

B. Interpretation of the Adaptive Framework

The adaptive framework was designed to support parameter adaptation, convergence monitoring, recovery, and autonomous stopping. The pilot results confirm that the software architecture can execute these modules, but they do not prove that the current parameterization is optimal. The adaptive method matched Max-Min rather than surpassing standard PSO in the larger scenario. One likely explanation is that the convergence-control thresholds or recovery policy may have made the swarm conservative, causing it to retain a strong heuristic-like solution instead of exploring enough alternative mappings. Another possibility is that the adaptive stop condition was configured too cautiously, which explains why the evaluation count remained identical to standard PSO.

These findings do not invalidate the framework; instead, they clarify its current status. A modular framework is valuable because the adaptation rule, diversity measure, recovery operator, and termination thresholds can be changed without rewriting the cloud model or result pipeline. Future experiments should test alternative inertia adaptation functions, larger recovery windows, load-aware task migration, and schedule-diversity thresholds normalized by task count.

C. Threats to Validity

The main internal-validity threat is that the pilot experiments are too limited to separate random variation from systematic performance differences. The small number of scenarios also limits external validity because cloud workloads may differ in task-size distribution, VM heterogeneity, price-speed correlation, deadlines, memory constraints, and dynamic arrival patterns. Construct validity is affected by the weighted composite objective because different

users may prefer different trade-offs among makespan, cost, and utilization. Finally, implementation validity depends on consistent schedule evaluation, reproducible random seeds, and fair evaluation budgets, all of which were addressed in the MATLAB architecture but still require larger-scale testing.

D. Required Refinements

The next refinement stage should execute a calibrated experimental design. Monitoring-only runs should record hypothetical convergence points without stopping the optimizer. The stability of the best schedule after each hypothetical stop should then be measured against the full-budget continuation. Termination thresholds should be selected from calibration scenarios and frozen before testing. Recovery should be compared with no-recovery and load-aware recovery variants. The final paper or thesis should include sensitivity analysis for swarm size, observation window, persistence count, recovery fraction, and objective weights.

CONCLUSION

This paper presented a modular convergence-aware adaptive PSO framework for resource-efficient cloud task scheduling and completed the initial manuscript with MATLAB pilot results. The framework models independent tasks and heterogeneous VMs, decodes particles into task-to-VM assignments, evaluates schedules through a normalized resource-aware fitness, and supports adaptive control, convergence monitoring, recovery, and autonomous termination. The MATLAB implementation was verified through unit tests and executed on 40-task/6-VM and 100-task/10-VM pilot scenarios.

The received results show that resource-aware scheduling strongly outperformed round robin. In the 40-task scenario, Max-Min, standard PSO, and CA-APSO all achieved mean fitness 0.12561 and makespan 29.159. In the 100-task scenario, standard PSO produced the best observed mean fitness of 0.054792 and a makespan of 41.605, slightly improving over Max-Min and CA-APSO. These outcomes verify the implementation and confirm the importance of resource-aware scheduling. However, the current adaptive configuration did not yet outperform standard PSO and did not reduce the number of evaluations. Therefore, superiority of the adaptive convergence-aware mechanism remains a future validation target rather than a demonstrated claim.

Future work will focus on threshold calibration, recovery redesign, larger workload scenarios, statistical validation over at least 30 independent runs, and comparison against additional PSO variants. The framework may also be extended to dynamic task arrivals, workflow scheduling, SLA-aware penalties, energy-aware scheduling, Pareto-based multi-objective optimization, and public workload traces. The main contribution of the current work is a verified and extensible MATLAB research platform that connects cloud schedule quality, adaptive PSO behavior, and optimizer-effort analysis within a single reproducible framework.

REFERENCES

1. J. Kennedy and R. C. Eberhart, “Particle swarm optimization,” in Proc. IEEE Int. Conf. Neural Networks, Perth, WA, Australia, 1995, vol. 4, pp. 1942–1948, doi: 10.1109/ICNN.1995.488968.
2. Y. Shi and R. C. Eberhart, “A modified particle swarm optimizer,” in Proc. IEEE Int. Conf. Evolutionary Computation, Anchorage, AK, USA, 1998, pp. 69–73, doi: 10.1109/ICEC.1998.699146.

3. M. Clerc and J. Kennedy, "The particle swarm—Explosion, stability, and convergence in a multidimensional complex space," IEEE Trans. Evol. Comput., vol. 6, no. 1, pp. 58–73, Feb. 2002, doi: 10.1109/4235.985692.
4. A. Nickabadi, M. M. Ebadzadeh, and R. Safabakhsh, "A novel particle swarm optimization algorithm with adaptive inertia weight," Appl. Soft Comput., vol. 11, no. 4, pp. 3658–3670, 2011, doi: 10.1016/j.asoc.2011.01.037.
5. S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in Proc. 24th IEEE Int. Conf. Advanced Information Networking and Applications, Perth, WA, Australia, 2010, pp. 400–407.
6. A. I. Awad, N. A. El-Hefnawy, and H. M. Abdel-Kader, "Enhanced particle swarm optimization for task scheduling in cloud computing environments," Procedia Comput. Sci., vol. 65, pp. 920–929, 2015.
7. X. Huang, C. Li, H. Chen, and D. An, "Task scheduling in cloud computing using particle swarm optimization with time varying inertia weight strategies," Cluster Comput., vol. 23, pp. 1137–1147, 2020, doi: 10.1007/s10586-019-02983-5.
8. S. Nabi, M. Ahmad, M. Ibrahim, and H. Hamam, "AdPSO: Adaptive PSO-based task scheduling approach for cloud computing," Sensors, vol. 22, no. 3, Art. no. 920, 2022, doi: 10.3390/s22030920.
9. S. A. Alsaidy, A. D. Abbood, and M. A. Sahib, "Heuristic initialization of PSO task scheduling algorithm in cloud computing," J. King Saud Univ.-Comput. Inf. Sci., vol. 34, no. 6, pp. 2370–2382, 2022, doi: 10.1016/j.jksuci.2020.11.002.
10. S. H. Anbarkhan, A. T. Sadiq, and M. A. Tawfeeq, "An enhanced PSO algorithm for scheduling workflow tasks in cloud computing," Electronics, vol. 12, no. 12, Art. no. 2580, 2023, doi: 10.3390/electronics12122580.
11. B. Bassimir, M. Schmitt, and R. Wanka, "Self-adaptive potential-based stopping criteria for particle swarm optimization with forced moves," Swarm Intell., vol. 14, pp. 285–311, 2020, doi: 10.1007/s11721-020-00185-z.
12. T. Shaqarin and B. R. Noack, "A fast-converging particle swarm optimization through targeted, position-mutated, elitism (PSO-TPME)," Int. J. Comput. Intell. Syst., vol. 16, Art. no. 6, 2023, doi: 10.1007/s44196-023-00183-z.
13. T. M. Shami et al., "Velocity pausing particle swarm optimization: A novel variant for global optimization," Neural Comput. Appl., vol. 35, pp. 9193–9223, 2023, doi: 10.1007/s00521-022-08179-0.
14. A. Ratnaweera, S. K. Halgamuge, and H. C. Watson, "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients," IEEE Trans. Evol. Comput., vol. 8, no. 3, pp. 240–255, Jun. 2004, doi: 10.1109/TEVC.2004.826071.
15. F. van den Bergh and A. P. Engelbrecht, "A study of particle swarm optimization particle trajectories," Inf. Sci., vol. 176, no. 8, pp. 937–971, 2006.
16. R. Poli, "Mean and variance of the sampling distribution of particle swarm optimizers during stagnation," IEEE Trans. Evol. Comput., vol. 13, no. 4, pp. 712–721, Aug. 2009.

17. Q. Yuan and G. Yin, "Analyzing convergence and rates of convergence of particle swarm optimization algorithms using stochastic approximation methods," IEEE Trans. Autom. Control, vol. 60, no. 7, pp. 1760–1773, 2015.

18. S. Jamali and R. Analoui, "Task scheduling in cloud computing using particle swarm optimization," Int. J. Comput. Appl., vol. 144, no. 6, pp. 32–38, 2016.

19. S. K. Patel and A. Singh, "Task scheduling in cloud computing using hybrid meta-heuristic: A review," arXiv:2201.09242, 2022.

20. D. Subramoney and C. N. Nyirenda, "A comparative evaluation of population-based optimization algorithms for workflow scheduling in cloud-fog environments," arXiv:2012.00176, 2020.

21. A. Chaitanya and G. Lakshmeeswari, "An adaptive load balancing using particle swarm optimization for cloud task scheduling," Int. J. Eng. Trends Technol., vol. 71, no. 9, pp. 36–45, 2023, doi: 10.14445/22315381/IJETT-V71I9P204.

22. F. Kaplan and A. Babalik, "Performance analysis of cloud computing task scheduling using metaheuristic algorithms in DDoS and normal environments," Electronics, vol. 14, no. 10, Art. no. 1988, 2025, doi: 10.3390/electronics14101988.

23. R. Prasad, A. Roy, and S. Kumari, "Enhancing cloud task scheduling using a hybrid particle swarm and grey wolf optimization approach," arXiv:2505.15171, 2025.

24. H. Huang, J. Qiu, and K. Riedl, "On the global convergence of particle swarm optimization methods," Appl. Math. Optim., vol. 88, Art. no. 30, 2023.

25. M. Gerwien, R. Voßwinkel, and H. Richter, "Convergence analysis of particle swarm optimization using stochastic Lyapunov functions and quantifier elimination," Int. J. Control, vol. 96, no. 1, pp. 1–14, 2023.

26. M. R. Bonyadi, "A theoretical guideline for designing an effective adaptive particle swarm," IEEE Trans. Evol. Comput., vol. 24, no. 1, pp. 57–68, Feb. 2020.

27. D. P. Rini, S. M. Shamsuddin, and S. S. Yuhaniz, "Particle swarm optimization: Technique, system and challenges," Int. J. Comput. Appl., vol. 14, no. 1, pp. 19–27, 2011.

28. D. Freitas, L. G. Lopes, and F. Morgado-Dias, "Particle swarm optimisation: A historical review up to the current developments," Entropy, vol. 22, no. 3, Art. no. 362, 2020, doi: 10.3390/e22030362.