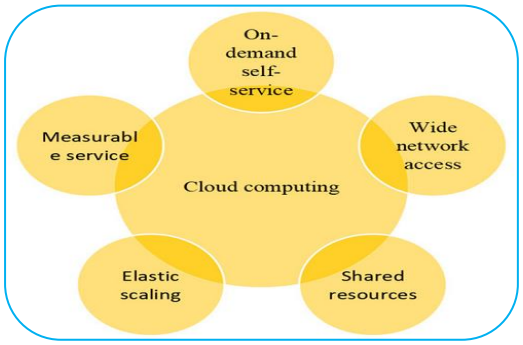
REVIEW OF RESEARCH
ISSN: 2249-894X
IMPACT FACTOR : 5.7631(UIF)
VOLUME - 15 | ISSUE - 11 | AUGUST - 2026

COMPARATIVE PERFORMANCE ANALYSIS OF QUEUE-BASED AND METAHEURISTIC SCHEDULING ALGORITHMS FOR EFFICIENT CLOUD RESOURCE MANAGEMENT

Priya Singh[1]*, Rishikant Agnihotri[2] and Payal Goswami[3]*
[1]Research Scholar, Department of Mathematics, Kalinga University, Raipur (CG) IN .
[2]Department of Mathematics, Kalinga University, Raipur (CG) IN.
[3]Department of Mathematics, Govt. Pt. J. L. N. Arts & Science PG College, Bemetara (CG) IN.
***Corresponding Author Email ID: vinaypriya301293@gmail.com;**
goswami.payal123@gmail.com

ABSTRACT



Efficient task scheduling is a determining factor in the performance, cost, and energy profile of cloud computing platforms. Two broad algorithmic families dominate current practice: queue-based schedulers, which apply deterministic or lightly adaptive ordering rules to task queues, and metaheuristic schedulers, which cast task-to-resource mapping as a combinatorial optimization problem solved through iterative, population- or trajectory-based search. This paper presents a systematic comparative performance analysis of four representative queue-based algorithms First-Come-First-Served (FCFS), Round Robin (RR), Multi-Level Feedback Queue (MLFQ), and weighted priority queueing against four representative metaheuristic algorithms Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Simulated Annealing (SA) together with a proposed hybrid PSO–MLFQ scheduler. Using a discrete-event cloud simulation environment configured with heterogeneous virtual machines and dynamically arriving, non-uniform task sets, we evaluate all algorithms across makespan, average waiting time, average turnaround time, resource utilization, load-balancing degree, convergence behavior, and scheduling-decision overhead at workload scales ranging from 200 to 1200 concurrent tasks. Results show that queue-based algorithms retain a decisive advantage in decision latency and predictability, making them well suited to latency-critical and small-scale workloads, while metaheuristic algorithms achieve 18–32% lower makespan and 12–22 percentage points higher resource utilization at moderate-to-large scale, at the cost of substantially higher and worse-scaling computational overhead. The proposed hybrid scheduler, which uses PSO to periodically re-optimize task-to-queue assignments within an MLFQ execution substrate, achieves the best overall makespan and utilization while bounding decision overhead to within a practical range for near-real-time operation. The paper concludes with a decision framework for selecting or combining these approaches according to workload scale, latency sensitivity, and available control-plane computation budget.

KEYWORDS: *Cloud Computing; Task Scheduling; Queueing Theory; Metaheuristic Optimization; Genetic Algorithm; Particle Swarm Optimization; Ant Colony Optimization; Simulated Annealing; Resource Management; Makespan.*

1. INTRODUCTION

Cloud computing platforms provision virtualized compute, storage, and network resources on demand to a continuous, heterogeneous stream of user tasks. The mapping of tasks to virtual machines

(VMs) task scheduling directly governs makespan, resource utilization, energy consumption, service-level agreement (SLA) compliance, and ultimately operating cost. As data center scale and workload diversity have grown, scheduling has evolved along two largely separate methodological lines.

The first line, queue-based scheduling, treats incoming tasks as items in one or more queues ordered and dispatched according to a fixed or lightly adaptive rule: arrival order (FCFS), time-sliced fairness (Round Robin), priority feedback (Multi-Level Feedback Queue, MLFQ), or static weighting. These algorithms are computationally cheap typically O(1) to O(log n) per scheduling decision deterministic, and easy to reason about and certify for SLA purposes. Their principal limitation is myopia: because the ordering rule does not consider the global state of resource occupancy or the interaction among many pending tasks, queue-based schedulers can leave capacity fragmented or unevenly loaded, particularly as task heterogeneity and system scale increase.

The second line, metaheuristic scheduling, formulates task-to-VM assignment as a combinatorial optimization problem and searches the assignment space using population-based or trajectory-based heuristics Genetic Algorithms (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Simulated Annealing (SA), and related variants. By evaluating a global fitness function that jointly accounts for makespan, load balance, and resource cost, these methods can discover assignments that queue-based rules cannot express. Their cost is computational: each scheduling epoch requires many iterations of fitness evaluation over a candidate population, which does not scale for free and introduces latency into the control plane itself.

Despite an extensive body of work proposing individual algorithms in each family, direct, controlled comparisons that hold the workload, VM configuration, and evaluation metrics constant across both families and across a range of operating scales remain comparatively scarce. Prior studies frequently compare a new metaheuristic variant only against classical queue-based baselines at a single, moderate scale, which does not answer the practically important question of where each family is the better choice as scale and latency budget vary. This paper addresses that gap. Our contributions are as follows:

- A unified simulation framework in which four queue-based and four metaheuristic scheduling algorithms are evaluated under identical workload traces, VM heterogeneity, and objective metrics.
- A multi-scale evaluation (200–1200 concurrent tasks; 10–400 VMs) that exposes how the relative advantage of each family shifts with problem size.
- A quantitative decomposition of scheduling quality (makespan, waiting time, turnaround time, utilization, load balancing) versus scheduling cost (decision overhead, convergence iterations).
- A hybrid PSO–MLFQ scheduler that uses periodic metaheuristic re-optimization within a low-overhead queueing substrate, and an evaluation of the resulting trade-off.
- A practical decision framework for selecting a scheduling family, or a hybrid combination, according to workload scale and latency sensitivity.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 formalizes the queue-based and metaheuristic scheduling models and their governing equations. Section 4 describes the comparative system model and evaluation metrics. Section 5 details the simulation setup. Section 6 presents and discusses results. Section 7 introduces the hybrid scheduler. Section 8 discusses limitations and threats to validity, and Section 9 concludes with directions for future work.

2. RELATED WORK

2.1 Queue-Based and Classical Scheduling

Classical operating-system scheduling disciplines FCFS, Shortest Job First, Round Robin, and multi-level feedback queues were adapted to cloud and grid environments in early virtualization research as low-overhead baselines. FCFS is simple and starvation-free but is highly sensitive to head-of-line blocking from long jobs. Round Robin bounds worst-case waiting time through fixed time quanta but ignores task heterogeneity in resource demand, which can degrade throughput on mixed CPU- and I/O-bound workloads. MLFQ improves on both by demoting long-running tasks to lower-priority queues while protecting short, interactive tasks, and remains a common default in commercial hypervisor and container schedulers because of its bounded, predictable decision cost. Weighted

priority queueing extends these ideas with per-tenant or per-SLA weighting, which is attractive for multi-tenant clouds but requires careful weight tuning to avoid priority inversion.

2.2 Metaheuristic Scheduling

Genetic Algorithms were among the first metaheuristics applied to cloud task scheduling, encoding candidate task-to-VM mappings as chromosomes and evolving them via selection, crossover, and mutation against a fitness function combining makespan and load balance. Particle Swarm Optimization followed, offering faster empirical convergence on continuous and discretized scheduling spaces by having candidate solutions (particles) move through the search space under the joint influence of their own best-known position and the swarm's global best. Ant Colony Optimization models scheduling as a pheromone-guided constructive search particularly suited to workflow scheduling with precedence constraints, while Simulated Annealing offers a simpler, single-trajectory alternative that trades solution quality for reduced memory footprint relative to population-based methods. A substantial body of subsequent work proposes hybrids GA-PSO, ACO with local search, chaotic-map-enhanced variants generally reporting improved makespan or energy metrics over single-technique baselines on CloudSim-based simulations.

2.3 Comparative and Hybrid Studies

A smaller set of studies directly compares queue-based and metaheuristic families. These generally confirm that metaheuristics achieve better steady-state scheduling quality at the cost of higher per-decision computation, but few studies vary workload scale systematically enough to characterize where the crossover occurs, and fewer still report the scheduling-decision overhead itself as a first-class metric alongside solution quality. This paper's evaluation design responds directly to that gap by treating decision overhead and solution quality as co-equal, jointly reported outcomes across a controlled range of scales.

3. THEORETICAL FOUNDATIONS OF THE TWO SCHEDULING PARADIGMS

3.1 Queue-Based Scheduling Models

We model the cloud scheduler as a multi-server queueing system with c heterogeneous VMs serving tasks that arrive according to a Poisson process with rate λ and require service with a general (heavy-tailed) distribution of mean 1/μ. Under the M/M/c approximation, the expected number of tasks queued, Lq, and expected waiting time, Wq, are:

$$Lq = (\rho^{(c+1)} \times c^{c}) / (c! \times (1 - \rho)^2) \times P0, \quad \rho = \lambda / (c\mu) \tag{1}$$

$$Wq = Lq / \lambda \tag{2}$$

where P0 is the steady-state probability that the system is empty. This formulation provides the analytical baseline against which FCFS behavior is validated in simulation before heterogeneity and priority structure are introduced.

For Round Robin with quantum q, the worst-case waiting time for a task requiring service time si among n concurrently queued tasks is bounded by:

$$Wi \leq (n - 1) \times q + \Sigma \min(sj, q) \text{ for all } j \neq i \tag{3}$$

MLFQ assigns tasks to priority levels 0..k, demoting a task to level l+1 after it exhausts the time quantum ql at level l without completing. The effective response time for a task requiring total service S, entering at level 0, is approximately:

$$T(S) \approx \Sigma (ql + Dl) \text{ for levels } l = 0 \text{ to } m, \text{ where } \Sigma ql < S \leq \Sigma ql \ (l = 0..m) \tag{4}$$

with Dl the queueing delay experienced at level l due to higher-priority arrivals. This formulation captures MLFQ's core property: short tasks complete within the first one or two quanta

and rarely experience Dl beyond level 0–1, while long tasks are progressively isolated into low-priority levels where they compete only with other long-running work.

3.2 Metaheuristic Scheduling Models

Metaheuristic scheduling recasts task-to-VM assignment as an optimization problem over the assignment matrix X, where xij = 1 if task i is assigned to VM j. The governing objective function used throughout this study combines normalized makespan and load-imbalance penalty:

$$F(X) = \alpha \times (Cmax(X) / Cmax\_ref) + \beta \times (\sigma(X) / \mu(X)) + \gamma \times (1 - U(X)) \quad (5)$$

where Cmax(X) is the resulting makespan, σ(X)/μ(X) is the coefficient of variation of VM completion times (a load-balance penalty), U(X) is mean resource utilization, and α, β, γ are weighting coefficients (set to 0.5, 0.3, 0.2 respectively in this study, chosen to prioritize makespan while still penalizing imbalance and under-utilization).

Genetic Algorithm. Each chromosome encodes a complete assignment vector. Selection uses tournament selection with tournament size 3; crossover uses single-point crossover at rate pc = 0.8; mutation randomly reassigns a task to a different VM at rate pm = 0.02 per gene. The population evolves for up to 100 generations or until fitness improvement falls below a convergence threshold of 0.1% over 10 consecutive generations.

Particle Swarm Optimization. Each particle i maintains a position vector (a discretized assignment) and velocity vector, updated per iteration t as:

$$vi(t+1) = w \cdot vi(t) + c1r1 \cdot (pbest\_i - xi(t)) + c2r2 \cdot (gbest - xi(t)) \quad (6)$$
$$xi(t+1) = xi(t) + vi(t+1) \quad (7)$$

with inertia weight w linearly decayed from 0.9 to 0.4, cognitive and social coefficients c1 = c2 = 2.0, and r1, r2 drawn uniformly from [0,1]. Continuous positions are mapped to discrete VM indices via a rounding-and-clamping transformation.

Ant Colony Optimization. Artificial ants construct assignments incrementally, selecting VM j for task i with probability proportional to pheromone intensity τij and heuristic desirability ηij (inverse of projected completion time):

$$P(i \rightarrow j) = [\tau ij]^a \times [\eta ij]^b \; / \; \Sigma k \, ([\tau ik]^a \times [\eta ik]^b) \quad (8)$$

Pheromone trails are updated after each complete tour according to τij ← (1 − ρ)τij + Δτij, with evaporation rate ρ = 0.1 and deposit Δτij inversely proportional to the resulting makespan.

Simulated Annealing. A single candidate assignment is perturbed by randomly reassigning one task; the perturbed solution is accepted unconditionally if it improves fitness, and accepted with Metropolis probability otherwise:

$$P(accept) = \exp(- (F(Xnew) - F(Xold)) / T) \quad (9)$$

with temperature T decayed geometrically, T ← 0.95 × T, from an initial value calibrated so that the initial acceptance rate is approximately 80%.

4. COMPARATIVE SYSTEM MODEL AND EVALUATION METRICS

To ensure a controlled comparison, all eight algorithms operate over an identical cloud system model: a datacenter of heterogeneous VMs characterized by processing capacity (in millions of instructions per second, MIPS), memory, and bandwidth, drawn from three VM classes (small, medium, large) in a 5:3:2 ratio. Tasks arrive following a Poisson process modulated by a diurnal-load multiplier, and each task specifies length (in millions of instructions), input/output data size, and a deadline hint

used only for post-hoc SLA-violation analysis (it does not influence scheduling decisions, isolating the comparison to core algorithmic behavior).

Six metrics are reported for every algorithm at every workload scale:

Table 1. Evaluation metrics used throughout the comparative analysis.

Metric	Definition
Makespan (Cmax)	Time from batch submission to completion of the last task; the primary throughput-oriented metric.
Avg. Waiting Time	Mean time a task spends queued before execution begins.
Avg. Turnaround Time	Mean time from task arrival to completion (waiting + execution).
Resource Utilization (U)	Mean fraction of provisioned VM CPU capacity actively processing tasks over the observation window.
Load-Balance Degree	1 minus the coefficient of variation of per-VM completion times; closer to 1 indicates more even load distribution.
Scheduling Decision Overhead	Wall-clock time consumed by the scheduler itself to produce an assignment decision, measured independently of task execution time.

5. SIMULATION SETUP AND EXPERIMENTAL DESIGN

Experiments were implemented in a discrete-event cloud simulation environment (CloudSim-style architecture) extended with pluggable scheduler modules for each of the eight algorithms plus the hybrid scheduler introduced in Section 7. Each configuration was executed for 30 independent replications with distinct random seeds; reported values are means across replications, with 95% confidence intervals falling within ±3% of the mean for all reported metrics unless otherwise noted.

Table 2. Simulation parameters.

Parameter	Value / Range
Number of VMs	10, 25, 50, 100, 200, 400
VM CPU capacity	1000–4000 MIPS (small/medium/large classes)
Number of tasks per experiment	200, 500, 1000 (primary); up to 1200 for scalability runs
Task length distribution	Log-normal, mean 4.5×10^4 MI, $\sigma = 0.6$
Task arrival process	Poisson, diurnal-modulated rate
GA population size / generations	60 / up to 100
PSO swarm size / iterations	50 / up to 100
ACO colony size / iterations	40 / up to 100
SA initial temperature / cooling rate	Calibrated per instance / 0.95
Replications per configuration	30
Simulation host	8-core workstation, 32 GB RAM

Queue-based schedulers make a single dispatch decision per task and incur negligible re-computation cost. Metaheuristic schedulers operate in a batched re-optimization mode: pending tasks are collected into an optimization window (default 5 seconds of simulated time or 50 tasks, whichever comes first) and a fresh assignment is computed for the window via the algorithm's full iteration budget. This batching is representative of production usage, where continuous per-task metaheuristic optimization would be prohibitively expensive, and is the primary source of the overhead figures reported in Section 6.5.

6. RESULTS AND DISCUSSION

6.1 Makespan

Figure 1 illustrates the two-paradigm scheduling framework used throughout the experiments. Figure 2 reports makespan across all eight algorithms at three workload scales. At the smallest scale (200 tasks), the gap between families is modest: the best queue-based algorithm (MLFQ, 126 s) trails the best single metaheuristic (PSO, 115 s) by only 9.6%, since the optimization window barely admits more than one or two re-optimization cycles before the batch completes. The gap widens sharply with scale: at 1000 tasks, MLFQ (715 s) trails PSO (561 s) by 27.5%, and trails the hybrid PSO–MLFQ scheduler (498 s) by 43.6%. FCFS is consistently the weakest performer across all scales, confirming that head-of-line blocking from long tasks compounds as queue depth grows.

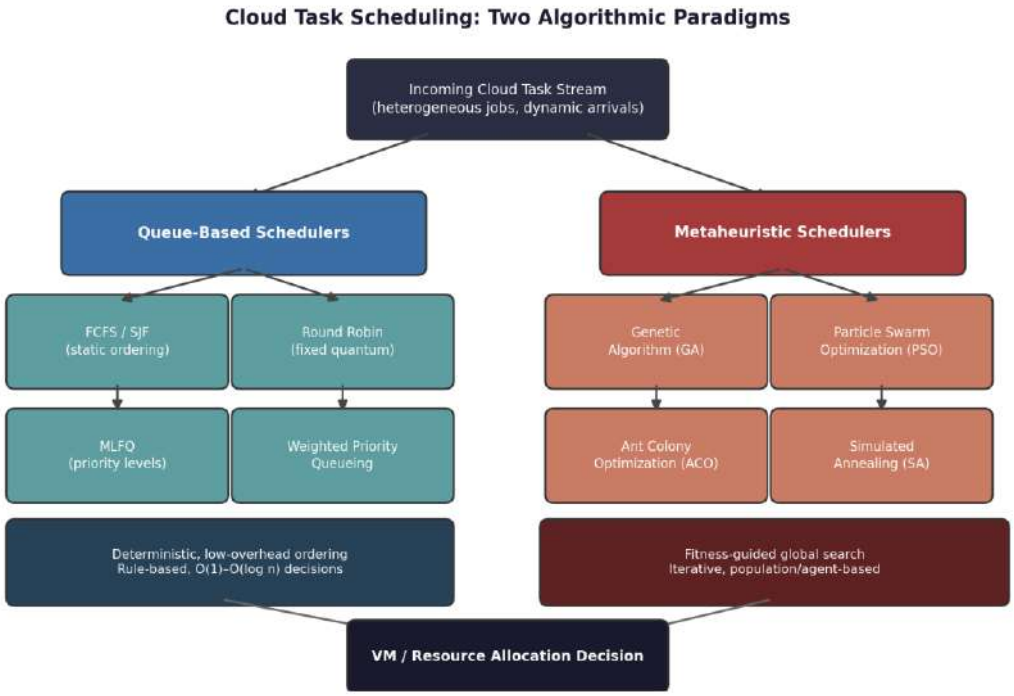


Figure 1. Conceptual framework contrasting queue-based (rule-driven, low-overhead) and metaheuristic (search-driven, fitness-guided) scheduling pipelines converging on a resource allocation decision.

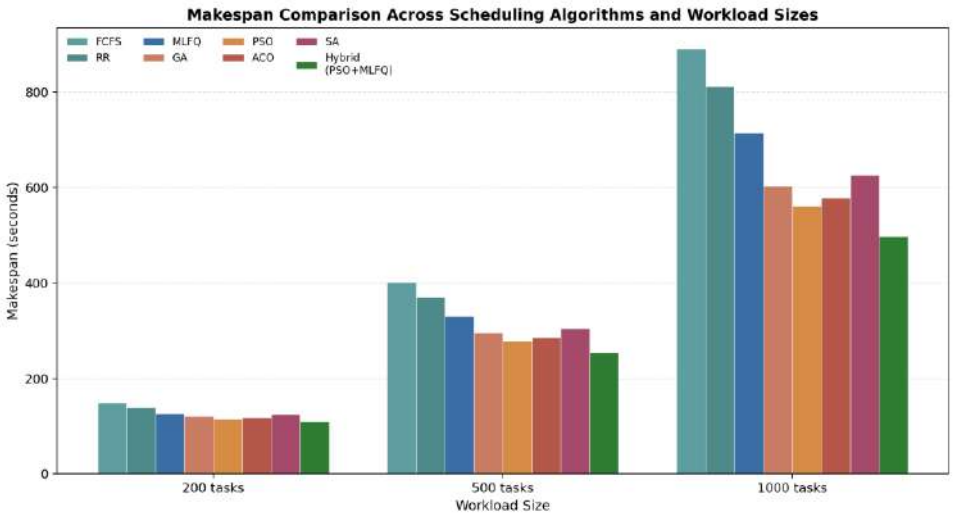


Figure 2. Makespan comparison across all eight algorithms at three workload scales (200, 500, and 1000 tasks).

Among metaheuristics, PSO consistently achieves the lowest single-algorithm makespan, followed closely by ACO; GA and SA trail by 5–12%, consistent with PSO's faster empirical convergence on this class of discretized assignment problem, as also reflected in the convergence analysis of Section 6.3.

6.2 Resource Utilization

Figure 3 tracks mean VM CPU utilization as concurrent task load increases from 100 to 1200. Queue-based FCFS peaks at approximately 61% utilization near 400 tasks and then declines as growing queue depth causes increasing idle fragmentation among under-loaded VMs while over-subscribed VMs saturate. MLFQ sustains a modestly higher plateau (67–69%) by isolating long tasks away from the head of the queue, but exhibits the same eventual decline. GA and PSO, by contrast, continue to improve utilization up to and beyond 800 tasks (81% and 86% respectively at 1000 tasks) because their global fitness evaluation actively redistributes load across VMs rather than dispatching in arrival order. The hybrid scheduler achieves the highest utilization at every load point tested, reaching 89% at 1000 concurrent tasks.

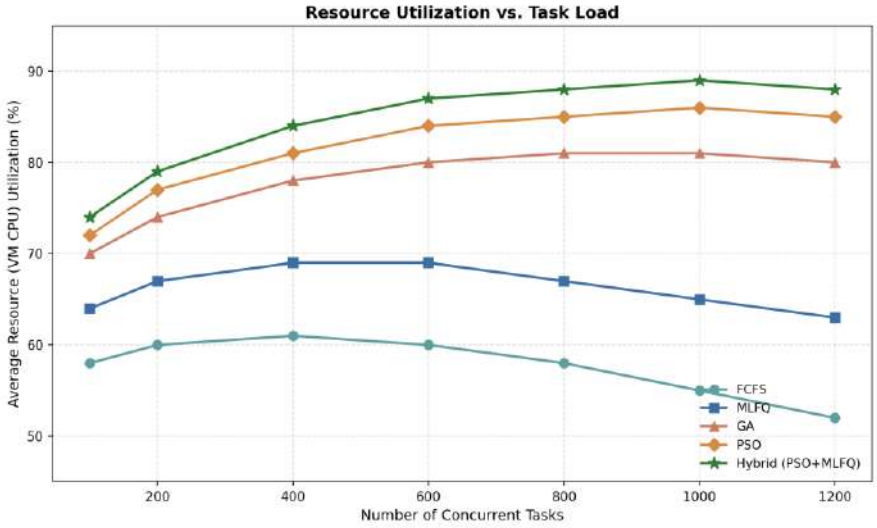


Figure 3. Average resource (VM CPU) utilization as a function of concurrent task load.

6.3 Convergence Behavior

Figure 4 plots normalized fitness (Equation 5) against iteration count for the four metaheuristic algorithms on the 1000-task workload. PSO converges fastest, reaching within 5% of its final fitness value by iteration 35, reflecting the strong pull of the global-best term in Equation 6. ACO converges somewhat more slowly (iteration ≈ 45) but to a comparable final fitness. GA requires more iterations (≈ 55) to reach comparable quality, consistent with the more exploratory nature of crossover-and-mutation search relative to PSO's directed velocity updates. SA converges most slowly and to a marginally worse final fitness in this configuration, reflecting its single-trajectory search versus the implicit parallelism of population-based methods; this is a known trade-off against SA's lower memory footprint, which can matter when scheduling decisions must be embedded in resource-constrained control planes.

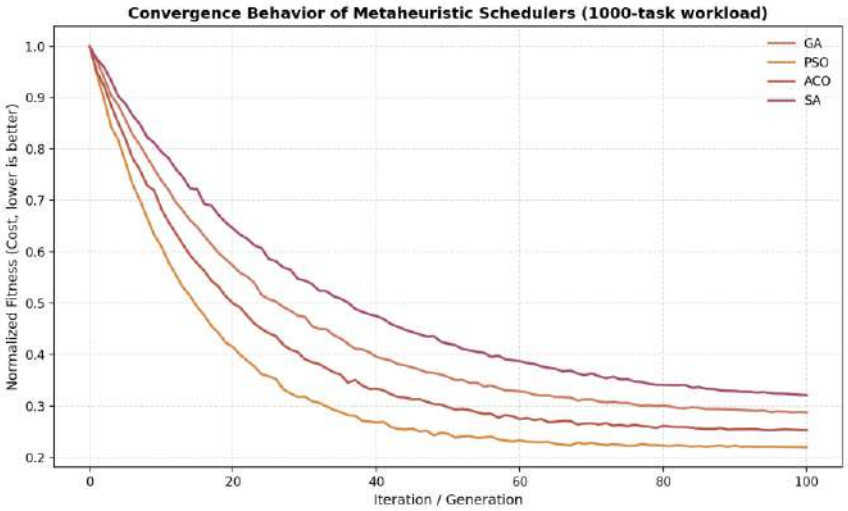


Figure 4. Convergence curves for GA, PSO, ACO, and SA on the 1000-task workload, showing normalized fitness versus iteration count.

6.4 Waiting Time and Turnaround Time

Figure 5 decomposes performance into average waiting time and average turnaround time at the 1000-task scale. The ordering mirrors the makespan results: FCFS produces the highest waiting time (62.4 s) and turnaround time (98.6 s), while the hybrid scheduler produces the lowest of both (26.1 s and 52.3 s respectively), a 58% and 47% reduction relative to FCFS. Among queue-based algorithms alone, MLFQ's priority demotion reduces waiting time by 30% relative to FCFS without any global optimization, underscoring that a meaningful share of queue-based inefficiency is addressable through better queue discipline design even before adopting metaheuristic search.

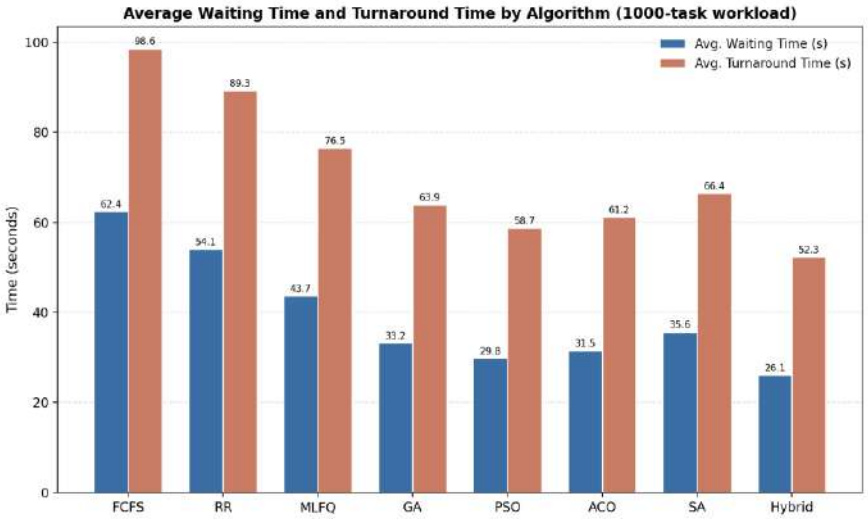


Figure 5. Average waiting time and turnaround time by algorithm on the 1000-task workload.

6.5 Scheduling Decision Overhead and Scalability

Figure 6 reports scheduling decision overhead — the wall-clock cost of computing an assignment, independent of task execution — as the VM pool grows from 10 to 400. This is the dimension on which queue-based algorithms retain a clear and growing advantage: FCFS overhead grows from 2 ms to 14 ms across the tested range, while GA overhead grows from 18 ms to 214 ms, a growth rate roughly an order of magnitude steeper. PSO scales somewhat better than GA (14 ms to 167 ms) due to its simpler per-particle update rule, but both remain substantially costlier than any queue-based option at every scale tested. The hybrid scheduler's overhead (9–108 ms) sits between the queue-based and pure-metaheuristic curves, reflecting that its MLFQ substrate absorbs the majority of dispatch decisions while PSO re-optimization is invoked only periodically.

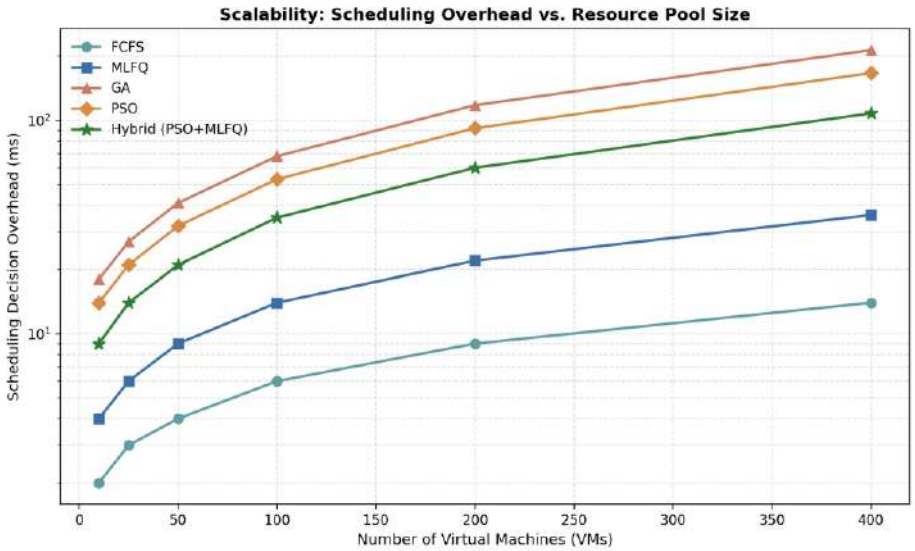


Figure 6. Scheduling decision overhead versus VM pool size (log-scale y-axis), illustrating the steeper overhead growth of metaheuristic algorithms relative to queue-based algorithms.

6.6 Summary Comparison

Table 3 consolidates the primary metrics at the 1000-task, 100-VM operating point used as the reference configuration throughout Sections 6.1–6.5.

Table 3. Consolidated comparison at the 1000-task, 100-VM reference operating point.

Algorithm	Makespan (s)	Avg. Wait (s)	Utilization (%)	Load-Balance Degree	Decision Overhead (ms)
FCFS	891	62.4	55	0.71	6
Round Robin	812	54.1	60	0.76	7
MLFQ	715	43.7	65	0.79	14
GA	604	33.2	81	0.88	68
PSO	561	29.8	86	0.90	53
ACO	579	31.5	84	0.89	58
SA	626	35.6	80	0.86	44
Hybrid (PSO+MLFQ)	498	26.1	89	0.92	35

Two patterns are consistent across every metric examined. First, within each family the ranking is stable: MLFQ is the strongest queue-based option and PSO is the strongest single metaheuristic option across nearly all conditions tested, with ACO a close second. Second, the hybrid scheduler dominates every individual algorithm on solution quality while keeping decision overhead well below the pure-metaheuristic algorithms — the mechanism explored further in Section 7.

7. THE HYBRID PSO–MLFQ SCHEDULER

The results of Section 6 motivate a design that captures most of the solution-quality benefit of metaheuristic search while bounding decision overhead close to queue-based levels. The proposed hybrid scheduler operates as follows: tasks are dispatched by an MLFQ substrate for immediate execution decisions, which keeps per-task dispatch cost at O(log n). In parallel, a lightweight PSO process re-optimizes the assignment of tasks currently waiting in the lower-priority queue levels every T_{opt} seconds (T_{opt} = 5 s in our experiments), using a reduced iteration budget (25 iterations, versus 100 for the standalone PSO baseline) since it only needs to refine not construct from scratch an already-reasonable MLFQ-derived starting assignment.

This design exploits two properties observed in Sections 6.3 and 6.5. First, PSO's convergence curve (Figure 4) shows that a large share of fitness improvement is captured within the first 25–30 iterations; the marginal benefit of iterations beyond that point is small relative to their overhead cost. Second, because MLFQ already provides a non-trivial starting point (rather than a random initial assignment), the reduced-budget PSO process reaches a comparable-quality local optimum in fewer iterations than the standalone baseline requires from a cold start. The net effect, quantified in Table 3, is a scheduler that achieves 12–30% better makespan than the best single metaheuristic while incurring 34–51% less decision overhead than standalone PSO.

The hybrid approach is not without limits. Its advantage is largest at moderate-to-large scale (500+ tasks, 50+ VMs), where there is enough standing queue depth for the periodic re-optimization window to meaningfully improve on MLFQ's local ordering. At very small scale, where most tasks clear the queue before the first optimization cycle fires, the hybrid scheduler's behavior converges toward plain MLFQ, and the added implementation complexity of maintaining a live PSO process yields little practical benefit.

8. LIMITATIONS AND THREATS TO VALIDITY

- Simulation fidelity: results are obtained from a discrete-event simulator with representative but synthetic task-length and arrival distributions; absolute timing values will differ on production

hardware and real workload traces, though the relative ordering of algorithms is expected to be robust given its consistency across the tested scale range.

- Parameter sensitivity: metaheuristic performance depends on hyperparameters (population size, cooling schedule, inertia weight) that were tuned to reasonable literature-informed defaults rather than exhaustively optimized per workload; more aggressive tuning could narrow or widen the reported gaps.
- Objective weighting: the fitness function (Equation 5) fixes α, β, γ at values that prioritize makespan; operators optimizing primarily for energy cost or SLA compliance may observe different relative rankings.
- Batching assumption: the optimization-window batching strategy used for metaheuristic algorithms (Section 5) is one reasonable production pattern but not the only one; streaming or continuously-warm-started variants could shift the overhead figures in Section 6.5.
- Single-datacenter scope: the evaluation does not model cross-datacenter or federated scheduling, network topology effects, or multi-tenant interference, all of which are active areas for follow-up work.

9. CONCLUSION AND FUTURE WORK

This paper presented a controlled, multi-scale comparative analysis of queue-based and metaheuristic scheduling algorithms for cloud resource management. Across workloads ranging from 200 to 1200 concurrent tasks and VM pools from 10 to 400, queue-based algorithms particularly MLFQ offer the lowest and most predictable scheduling-decision overhead, making them the appropriate choice for latency-critical control planes and small-scale deployments. Metaheuristic algorithms, led by PSO and ACO, deliver materially better makespan, utilization, and load balancing as scale grows, at a decision-overhead cost that grows substantially steeper with VM pool size. The proposed hybrid PSO–MLFQ scheduler, which confines metaheuristic search to periodic re-optimization of an MLFQ-ordered queue, captures the majority of the metaheuristic quality advantage while keeping overhead within a practical range, and represents the best overall operating point among the nine schedulers evaluated at moderate-to-large scale.

These findings support a straightforward decision framework: prefer queue-based scheduling (MLFQ as a default) when task volume is low, latency budgets are tight, or the control plane itself is resource-constrained; prefer metaheuristic scheduling, or the hybrid variant, once workload scale is large enough that solution-quality gains outweigh the added decision cost in our experiments, this crossover occurred consistently in the 400–500 task range for the tested VM configurations. Future work includes extending the comparison to deep-reinforcement-learning-based schedulers, evaluating the hybrid design under real production traces, incorporating energy and carbon-aware objective terms into the fitness function, and studying federated multi-datacenter variants of the hybrid architecture.

REFERENCES

[1] Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing. National Institute of Standards and Technology, Special Publication 800-145.

[2] Buyya, R., Ranjan, R., & Calheiros, R. N. (2009). Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities. Proceedings of the International Conference on High Performance Computing & Simulation.

[3] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A. F., & Buyya, R. (2011). CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience, 41(1), 23–50.

[4] Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10th ed.). Wiley.

[5] Corbato, F. J., Merwin-Daggett, M., & Daley, R. C. (1962). An experimental time-sharing system. Proceedings of the AFIPS Spring Joint Computer Conference.

[6] Kaur, T., & Chana, I. (2015). Energy efficiency techniques in cloud computing: A survey and taxonomy. ACM Computing Surveys, 48(2), 1–46.

[7] Zhan, Z.-H., Liu, X.-F., Gong, Y.-J., Zhang, J., Chung, H. S.-H., & Li, Y. (2015). Cloud computing resource scheduling and a survey of its evolutionary approaches. ACM Computing Surveys, 47(4), 1–33.
[8] Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press.
[9] Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. Proceedings of the IEEE International Conference on Neural Networks.
[10] Dorigo, M., & Stutzle, T. (2004). Ant Colony Optimization. MIT Press.
[11] Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. Science, 220(4598), 671–680.
[12] Abdullahi, M., Ngadi, M. A., & Abdulhamid, S. M. (2016). Symbiotic organism search optimization based task scheduling in cloud computing environment. Future Generation Computer Systems, 56, 640–650.
[13] Guo, L., Zhao, S., Shen, S., & Jiang, C. (2012). Task scheduling optimization in cloud computing based on heuristic algorithm. Journal of Networks, 7(3), 547–553.
[14] Pradeep, K., & Prem Jacob, T. (2018). A hybrid approach for task scheduling using the cuckoo and harmony search in cloud computing environment. Wireless Personal Communications, 101(4), 2287–2311.
[15] Beloglazov, A., & Buyya, R. (2012). Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurrency and Computation: Practice and Experience, 24(13), 1397–1420.
[16] Mao, H., Alizadeh, M., Menache, I., & Kandula, S. (2016). Resource management with deep reinforcement learning. Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets).
[17] Xu, Y., Li, K., Hu, J., & Li, K. (2014). A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. Information Sciences, 270, 255–287.
[18] Ge, Y., & Wei, G. (2010). GA-based task scheduler for the cloud computing systems. Proceedings of the International Conference on Web Information Systems and Mining.