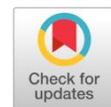


Optimizing priority scheduling in Hadoop for resource utilization using quantum particle swarm optimization technique



Bhavana Potli ^{a,1,*}, Shashikumar Dandinashivara Revanna ^{b,2}

^a Cambridge Institute of Technology, Bengaluru, Visvesvaraya Technological University, Belagavi, 590018, India

^b Sai Vidya Institute of Technology, Bengaluru, Visvesvaraya Technological University, Belagavi, 590018, India

¹ potli.bhavana13@gmail.com; ² shashikumardr99@gmail.com

* corresponding author

ARTICLE INFO

Article history

Received November 26, 2025

Revised April 28, 2026

Accepted April 28, 2026

Available online August 31, 2026

Keywords

Data locality

Distributed scheduling

Hadoop YARN

Priority-aware optimization

Resource management

ABSTRACT

Efficient resource scheduling in Hadoop remains a challenging problem due to the presence of diverse workloads and varying priority requirements in cluster environments. Traditional YARN schedulers such as FIFO, Fair, and Capacity often struggle to balance data locality, responsiveness, fairness, and priority handling, which can increase waiting times and reduce resource utilization. To address these limitations, this study proposes a Quantum-Inspired Priority Scheduler (QIPS) that integrates a Quantum-behaved Particle Swarm Optimization (QPSO) mechanism within the YARN ResourceManager to enhance job-node assignment decisions. The proposed scheduler considers multiple performance criteria, including latency, resource utilization, data locality, and priority awareness, enabling more adaptive and balanced scheduling under dynamic workload conditions. A hybrid implementation combining Java and Python is developed, where YARN handles job execution while the QPSO module performs optimization. Experimental evaluation on a multi-node Hadoop 3.3.4 cluster using synthetic workloads shows that QIPS effectively reduces deadline penalties and improves data locality, while maintaining competitive performance across other scheduling metrics. These findings indicate that quantum-inspired optimization offers a promising direction for achieving efficient and balanced resource scheduling in distributed systems.



© 2026 The Author(s).

This is an open access article under the [CC-BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



1. Introduction

Hadoop has emerged as a leading framework for processing large-scale datasets due to its distributed, scalable, and fault-tolerant design. It integrates the Hadoop Distributed File System (HDFS) for reliable data storage with the MapReduce model for efficient parallel computation. Effective resource management is vital in such systems to ensure high throughput and balanced workload distribution. YARN (Yet Another Resource Negotiator) plays this role by dynamically allocating CPU, memory, and storage based on system availability and job priorities, improving overall cluster performance. However, the default YARN schedulers, FIFO and Fair, struggle to maintain both priority awareness and optimal resource use in latency-sensitive environments. Their static configurations and limited adaptability often lead to inefficient allocation, with critical tasks waiting while low-priority jobs consume valuable resources. Such issues reduce throughput and scalability, particularly in heterogeneous clusters. Recent studies have emphasized the need for adaptive, optimization-driven schedulers to overcome these constraints [1]. Quantum-inspired algorithms, which incorporate concepts from quantum mechanics such as probabilistic state transitions and superposition, offer a promising solution. These algorithms,

particularly Quantum-behaved Particle Swarm Optimization (QPSO), have demonstrated superior global search capabilities and faster convergence in complex scheduling problems [2]. Building on this potential, the proposed framework introduces a QPSO-based scheduler within the YARN ResourceManager to dynamically optimize job execution order and resource allocation. The scheduler constantly analyses system conditions and job priorities to improve scheduling decisions in real time. Experimental results show that it reduces average job completion time, increases resource utilization, and improves overall throughput across various workloads. Such quantum-inspired methods are increasingly being adopted to meet the performance demands of modern distributed and Industry 4.0 environments [3].

Several studies have examined resource scheduling in Hadoop, focusing on improving job placement, locality, and execution efficiency. Earlier work explored cache locality enhancement, workflow optimization, heterogeneity-aware scheduling, energy-aware policies, and metaheuristic-based task allocation, resulting in significant improvements to MapReduce and YARN environments [4]–[16]. While these approaches enhanced specific aspects of scheduling, most lacked integrated support for priority awareness, multi-objective optimization, or adaptive decision-making under rapidly changing workloads. More recent investigations have advanced Hadoop optimization through predictive modeling, multi-criteria selection mechanisms, frugal scheduling strategies, and distributed deep-learning pipelines. These studies demonstrated that enhanced data management, resource sharing, and container coordination can significantly increase throughput and data locality in modern heterogeneous clusters [17]–[23]. Such research highlights the need for schedulers that can react dynamically to fluctuations in job characteristics, data volumes, and cluster configurations. Parallel developments in cloud, edge, and IoT scheduling have also contributed to the evolution of large-scale resource management. Deadline-constrained task allocation, reinforcement learning-based scheduling, device-edge-cloud cooperation, and multi-objective optimization frameworks have shown strong improvements in responsiveness, fairness, and execution stability across diverse environments [24]–[28]. These trends illustrate the growing demand for intelligent, adaptive scheduling techniques that account for both system- and workload level constraints. Complementary studies have explored advanced grid scheduling, AI-driven task orchestration, heterogeneity handling, user aware edge scheduling, and scalable data-processing pipelines. These approaches provide additional evidence that incorporating context aware policies, multi-objective heuristics, and structured optimization can substantially enhance performance in distributed systems [29]–[36]. In domain-specific applications, YARN-based parallel processing frameworks have been successfully used to improve locality and computational throughput for large remote-sensing datasets, further validating the need for optimized resource-management strategies [37].

While a wide range of scheduling approaches have been proposed across Hadoop, cloud, and distributed systems, most existing methods address individual aspects such as locality, priority, or optimization in isolation. In contrast, methods closely related to the proposed approach include metaheuristic-based schedulers and multi-objective optimization frameworks in Hadoop/YARN environments. However, these approaches generally lack explicit task node encoding, unified multi-objective modeling, and effective adaptation of continuous optimization techniques to discrete scheduling decisions. The proposed QIPS framework addresses these gaps by integrating structured encoding, multi-objective optimization, and discrete adaptation within the YARN scheduling architecture. Existing Hadoop scheduling approaches lack explicit job task node encoding, effective multi-objective formulations, and robust constraint handling. They also provide limited support for bridging continuous optimization with discrete scheduling decisions and lack sufficient sensitivity analysis. The proposed QIPS addresses these limitations by introducing a structured particle representation and a unified multi-objective framework for priority-aware and locality-driven scheduling. It adapts QPSO for discrete YARN environments with constraint-aware mechanisms and is validated through sensitivity and workload-based analysis. Overall, the approach demonstrates improved scheduling balance and practical applicability under dynamic workload conditions. The main contributions of this study are as follows: (1) Developed a quantum-inspired scheduling approach that prioritizes jobs while improving data locality and reducing deadline violations in Hadoop YARN; (2) Designed a multi-objective optimization framework that balances latency, resource utilization, and

priority handling for efficient scheduling decisions; (3) Implemented a hybrid Java–Python framework to integrate QPSO-based optimization with YARN, enabling dynamic and real-time scheduling evaluation. Evaluated the model on real and synthetic workloads, showing significant improvements in speed, throughput, and utilization over standard schedulers. The rest of this paper is organized as follows. Section 2 explains the proposed methodology and framework design, the implementation setup, and outlines the experimental environment. Section 3 discusses the results and performance analysis. Finally, Section 4 concludes the paper and highlights future research directions.

2. Method

The proposed Quantum-Inspired Priority Scheduler (QIPS) is integrated into YARN's ResourceManager and works with the ApplicationMaster to allocate containers based on job priority, data locality, and current cluster conditions. As shown in Fig. 1, incoming jobs are first placed in a queue, where they are assigned priority and deadline tags, and then admitted to ensure fairness and avoid overload. For each selected job, the system collects relevant information such as input size, runtime history, and resource demand. Using this information, the Particle Generator creates candidate job task–node mappings, where each particle represents a complete scheduling plan and defines the search space for optimization. Each particle represents a complete scheduling mapping as defined in (1).

$$P = \{j, T, n\} \quad (1)$$

where j denotes the job index, T represents the task within the job, and n indicates the assigned node. The encoding is implemented as a discrete vector, where each dimension corresponds to a task and the associated value specifies its node assignment. This explicit representation distinguishes QIPS from prior approaches, in which task-to-node mappings are often implicit or poorly defined.

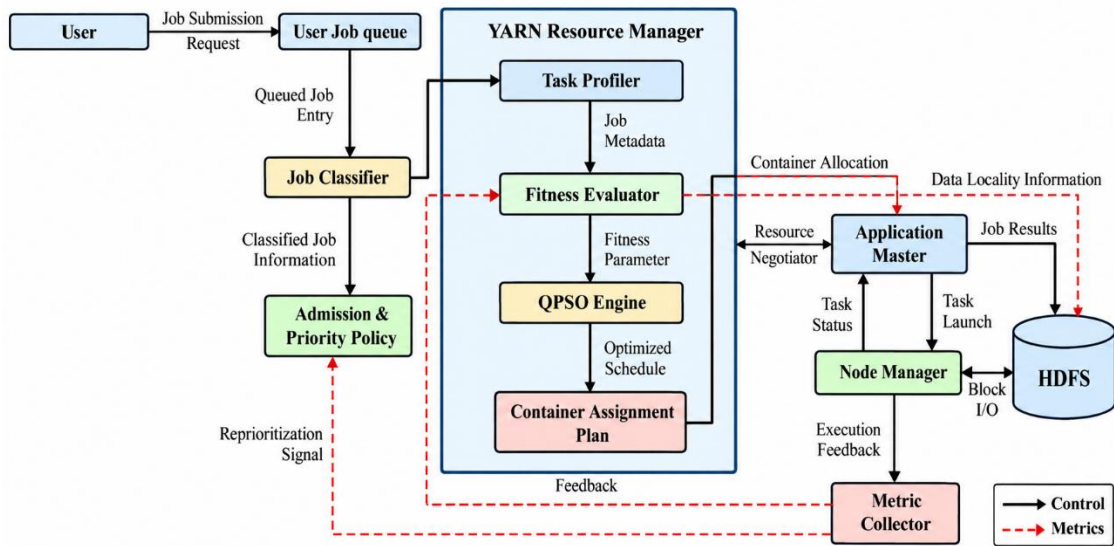


Fig. 1. System Architecture.

2.1. Multi-Objective Fitness Evaluation

Each particle is evaluated using a multi-objective fitness function that thoughtfully balances execution efficiency, responsiveness, and resource utilization, as shown in (2).

$$F(P_i) = w_1 T_{makespan} + w_2 T_{latency} + w_3 P_{deadline} - w_4 R_{usage} - w_5 L_{data} - w_6 S_{priority} \quad (2)$$

Here, $T_{makespan}$ denotes total job completion time, $T_{latency}$ the average waiting time, and $P_{deadline}$ the penalty for missed deadlines. R_{usage} represents resource utilization, L_{data} measures the proportion of data-local tasks, and $S_{priority}$ expresses the priority score. The first three terms are minimized, whereas

the last three are maximized (through their negative signs). Weights $w_1 - w_6$ are tunable to match system policy. The weights in the multi-objective function are empirically tuned to reflect the relative importance of different scheduling objectives. A sensitivity analysis was conducted by varying the weights in the range of 1 to 3 to study their impact on scheduling performance. It was observed that values below 1 reduce the influence of an objective, while values above 3 tend to overemphasize specific objectives, leading to biased or unstable outcomes. Based on these observations, weights within this range were selected to ensure balanced contributions across all objectives. In the experiments, the weight vector was set to $W = [1.0, 1.0, 2.5, 1.0, 1.2, 2.0]$, assigning slightly higher importance to deadline-related objectives. A Data-Locality Estimator retrieves block location information from HDFS to guide locality-aware scheduling. Simultaneously, a Metrics Collector gathers runtime statistics, such as queue length, container utilization, latency, and the deadline-miss ratio. These metrics provide feedback to the Fitness Evaluator and Admission Policy, forming a closed, adaptive loop. The proposed approach incorporates basic constraint handling to ensure feasible scheduling decisions. Resource capacity limits, node suitability, and deadline considerations are accounted for during task assignment. Infeasible mappings are implicitly avoided through controlled particle generation and validation during scheduling. This ensures that the generated schedules remain practical and aligned with system constraints.

2.2. QPSO Engine

Quantum behaved Particle Swarm Optimization (QPSO) replaces traditional velocity updates with a probabilistic, quantum inspired search mechanism. Metaheuristic and swarm based techniques demonstrate strong global exploration and stable convergence in multi objective scheduling for distributed systems, and recent advancements in quantum inspired and enhanced PSO variants further reinforce their effectiveness in high-dimensional optimization [38]–[42]. These characteristics make QPSO particularly suitable for high-dimensional scheduling spaces, such as Hadoop YARN, where solution quality depends on balancing locality, priority, and resource constraints as shown in (3) and (4).

$$x_{ij}(t+1) = p_{ij}(t) \pm \beta^* |m_j(t) - x_{ij}(t)| *_{ln} \left(\frac{1}{u}\right) \quad (3)$$

$$u \sim U(0,1) \quad (4)$$

where i is particle index, j is dimension index, t is the time, $x_{ij}(t)$ is the current position, $x_{ij}(t+1)$ is the updated position, $p_{ij}(t)$ is the personal best of particle i , $m_j(t)$ is the mean best position across the swarm, β controls exploration and exploitation, u is a uniform random variable, and $U(0,1)$ is uniform distribution, a distribution from which u is drawn (between 0 and 1) at each iteration to introduce stochastic behavior in the update process. The swarm optimizes a priority vector, which is decoded into a feasible scheduling solution. A Random-Keys with Deterministic Decoder approach is used to map continuous optimization outputs to discrete scheduling decisions. Each particle is represented by a position vector $x_i \in \mathbb{R}^n$, where n denotes the number of tasks and each dimension corresponds to a priority value. A permutation π is obtained using the argsort function, which orders tasks by priority. The decoder then assigns tasks sequentially to suitable nodes based on feasibility and system conditions. This probabilistic model enhances global search and mitigates premature convergence. For comparison, the traditional PSO uses velocity-based updates, given by (5) and (6).

$$v_{ij}(t+1) = \omega v_{ij}(t) + c_1 r_1 (p_{ij} - x_{ij}(t)) + c_2 r_2 (g_1 - x_{ij}(t)) \quad (5)$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \quad (6)$$

where ω is the inertia weight, $v_{ij}(t)$ is the Current velocity of the particles, $v_{ij}(t+1)$ is the updated velocity of the particles, c_1 and c_2 are cognitive and social coefficients, and $r_1, r_2 \in [0,1]$ are random numbers. By eliminating velocity, QPSO simplifies parameter tuning and achieves faster convergence in high-dimensional, discrete scheduling spaces. Quantum-behaved particle swarm optimization (QPSO) operates in a continuous search space and therefore requires adaptation for discrete scheduling

applications. In this work, continuous particle updates are transformed using a sigmoid function, followed by ranking-based assignment to generate valid discrete schedules. This transformation enables the optimization process to produce practical and implementable scheduling decisions within the target system.

2.3. Quantum-Inspired Priority Scheduling (QIPS) Algorithm

The Quantum-Inspired Priority Scheduling (QIPS) algorithm applies principles of Quantum-behaved Particle Swarm Optimization (QPSO) to determine an efficient job execution order based on multiple scheduling criteria. Each particle represents a candidate scheduling solution, which is evaluated using a multi-objective fitness function that incorporates priority, latency, and resource constraints. Particle positions are iteratively updated following QPSO dynamics to explore the search space and identify improved scheduling solutions. Algorithm 1 presents the detailed steps of the proposed approach. The final output is an optimized scheduling plan that balances resource utilization and priority-aware execution.

2.4. Execution and Feedback Adaptation

The QPSO engine identifies the optimal particle, which represents the final container assignment plan, and passes it to the ResourceManager for execution in coordination with the ApplicationMaster. Runtime metrics collected from NodeManagers and HDFS monitor scheduling performance under varying workloads. This integration ensures effective execution and supports consistent scheduling behavior within the YARN framework. Fig. 2 presents the QIPS algorithm.

```

Algorithm 1: QIPS algorithm
1.  Input:  $J \leftarrow$  Set of jobs/tasks
2.   $N \leftarrow$  Set of cluster nodes
3.  Resource capacities, priority, deadlines, and locality information of jobs
4.  QPSO parameters ( $S, T, \beta$ )
5.  Weight vector  $W$ 
6.  Output: Optimized scheduling plan
7.  Initialize a swarm of  $S$  particles  $x_1$ 
8.  For each particle  $x_1$  do
9.      Encode  $x_i$  using random keys
10.   Initialize  $pbest_i$ 
11. End For
12. Determine the global best  $gbest$ 
13. For  $t = 1$  to  $T$  do
14.   For each particle  $x_1$  do
15.     Convert  $x_1$  to permutation  $\pi_1$  using argsort
16.     For each task in  $\pi_1$  do
17.       Select a feasible node based on locality, load, and resource availability
18.       Assign task to node
19.     End For
20.   Check scheduling constraints
21.   Apply the repair mechanism if violations occur
22.   Evaluate fitness using weighted objectives
23.   If fitness ( $x_1$ ) is better than  $pbest_i$  then
24.      $pbest_i \leftarrow x_1$ 
25.   End If
26. End For
27. Update  $gbest$  from all  $pbest_i$ 
28. Compute  $mbest$ 
29. For each particle  $x_1$  do
30.   Update  $x_1$  using QPSO rule
31. End For
32. If the convergence criterion is satisfied, or the maximum iteration  $T$  is reached, then
33.   Break
34. End If
35. End For
36. Return the schedule corresponding to  $gbest$ 

```

Fig. 2. QIPS algorithm

2.5. Implementation

The proposed Quantum-Inspired Priority Scheduler (QIPS) was implemented on a Hadoop 3.3.4 cluster using a modular design. Fig. 3 illustrates the implementation dashboard. It integrates with the YARN ResourceManager via a custom Java plugin that handles job profiling and container allocation, while a Python-based QPSO service performs the optimization via a REST API. During scheduling events, the scheduler sends job details to the QPSO service, which returns an optimized execution order. The scheduler then uses this to assign containers and coordinate task execution with the ApplicationMaster and NodeManagers. Additionally, a metrics module tracks system performance, and a data locality component uses HDFS information to improve task placement. This setup enables efficient and priority-aware scheduling in a dynamic Hadoop environment.

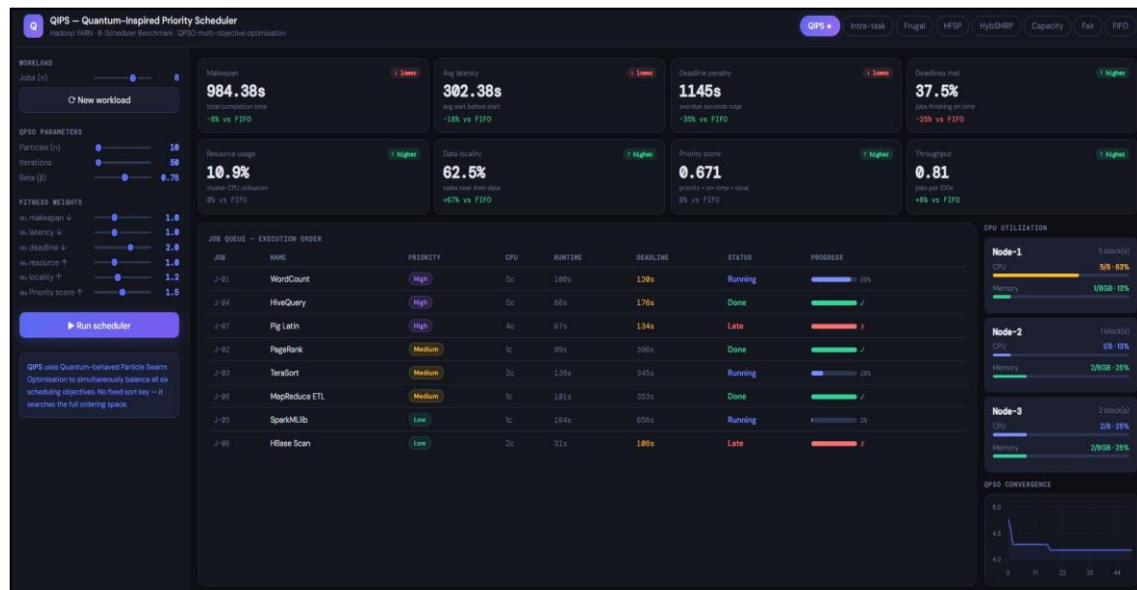


Fig. 3. Implementation dashboard used for monitoring scheduling behavior and system metrics (Random Screenshot)

2.6. Experimental Setup

The proposed approach was evaluated on a heterogeneous Hadoop 3.3.4 cluster consisting of one NameNode (16 GB RAM, 4 CPU cores) and three DataNodes (each with 8 GB RAM and 4 CPU cores). This configuration provides a realistic distributed environment for assessing scheduling performance. Synthetic workloads were generated to represent diverse job characteristics, including variations in priority, execution time, and resource requirements. In the experiments, a workload consisting of 8 jobs was used, with the number of jobs configurable to support different workload scenarios. Job instances were generated using the built-in YARN workload generation utilities, which assign randomized characteristics to each job, including priority, execution time, and resource requirements. This setup enables the simulation of heterogeneous and dynamic workload conditions. The performance of the proposed QIPS scheduler was evaluated by comparing it with standard Hadoop schedulers (FIFO, Fair, and Capacity) and with advanced approaches (HybSMRP, HFSP, Frugal_conf, and Intra-task Localization). Similar modular, service-oriented scheduling frameworks have demonstrated adaptability in heterogeneous environments, while recent studies on YARN scheduler design support the use of hybrid optimization techniques to improve performance under comparable workload conditions [43]–[47]. The evaluation focuses on key metrics such as latency, deadline adherence, resource utilization, data locality, and throughput to analyze scheduling effectiveness. Although the experiments are conducted on a small-scale cluster, the proposed approach is designed to be extendable to larger clusters, as the scheduling mechanism operates at the ResourceManager level. However, large-scale validation under heavier workloads is part of future work.

2.7. Evaluation Metrics

Makespan ($T_{makespan}$) is defined as the total time required to complete all submitted jobs, as shown in Formula (7).

$$T_{makespan} = \max_{1 \leq j \leq n}(t_j) \quad (7)$$

where t_j is the completion time of the job j , $\max(t_j)$ is the maximum operator, i.e., the largest value among all job completion times, with a condition that indicates all jobs from 1 to n , n is the number of jobs/tasks. In continuous time, Makespan can also be represented as the integral of average task progress over the execution window, $t_j(t)$, as shown in Formula (8).

$$T_{makespan} = \int_0^T \left(\frac{1}{n} \sum_{j=1}^n \frac{dt_j(t)}{dt} \right) dt \quad (8)$$

The latency metric $T_{Latency}$ given by (9) and (10) measures the average delay between a job's submission and the start of its first task.

$$T_{Latency} = \frac{1}{n} \sum_{j=1}^n (t_{start,j} - t_{submit,j}) \quad (9)$$

where, $t_{start,j}$ is the time at which the job j begins execution, and $t_{submit,j}$ is the time at which the job j was submitted. When considered as a continuous-time signal, latency becomes

$$T_{Latency} = \frac{1}{n} \int_0^T \left(\sum_{j=1}^n \delta(t - t_{start,j}) \cdot (t - t_{submit,j}) \right) dt \quad (10)$$

where, $\delta(t - t_{start,j})$ denotes the Dirac delta function used to capture the start time of the task j within the continuous-time formulation, $(t - t_{submit,j})$ represents the waiting time of the task j , and when combined with the Dirac delta function, it captures the latency at the exact task start time. The deadline penalty $P_{deadline}$ in (11) and (12) captures the delay beyond each job's deadline D_j . No penalty is applied if a job finishes early; otherwise, the penalty equals the delay.

$$P_{deadline} = \sum_{j=1}^n \max(0, t_j - D_j) \quad (11)$$

where, $\max(0, t_j - D_j)$ represents the delay beyond the deadline for the job j assigning a zero penalty to jobs that are completed on time. This penalty can also be written using the ReLU (Rectified Linear Unit) function:

$$P_{deadline} = \sum_{j=1}^n \text{ReLU}(t_j - D_j) \quad (12)$$

This term is maximized during optimization, and its negative is minimized in the fitness function to align with the overall minimization goal. Resource usage R_{usage} in Formula (13), the proportion of resources used over time relative to the cluster's total capacity is represented.

$$R_{usage} = \frac{1}{T} \int_0^T \left(\frac{\sum_{j=1}^n U_j(t)}{C(t)} \right) dt \quad (13)$$

where, $U_j(t)$ represents the resource usage of the task j at time t , $C(t)$ denotes the total available cluster capacity at time; the factor $\frac{1}{T}$ normalizes the integral over the observation period to give the

average resource utilization. Data locality L_{data} , Formula (14), measures the proportion of tasks that access local data.

$$L_{data} = \frac{1}{n} \sum_{j=1}^n l_j \quad (14)$$

where, l_j is a binary locality indicator for the task j . It equals 1 if the task runs on a node that contains its input data block, and 0 otherwise. Priority satisfaction $S_{priority}$ as shown in (15) and (16), it evaluates how effectively the scheduler favors high-priority jobs.

$$S_{priority} = \frac{\sum_{j=1}^n p_j \cdot s_j}{\sum_{j=1}^n p_j} \quad (15)$$

Here, p_j denotes the priority weight of the task j , and s_j denotes the scheduling satisfaction indicator or completion score of the task j . This can also be modeled in continuous time as

$$S_{priority} = \frac{1}{T} \int_0^T \sum_{j=1}^n p_j(t) \cdot \frac{ds_j(t)}{dt} dt \quad (16)$$

Here, $p_j(t)$ represents the time-varying priority weight, and $\frac{ds_j(t)}{dt}$ represents the rate at which scheduling satisfaction is achieved over time. Overall, these metrics provide a solid foundation for assessing the effectiveness of the proposed QIPS scheduler, highlighting improvements in throughput, responsiveness, fairness, and resource utilization compared to baseline Hadoop schedulers.

3. Results and Discussion

To evaluate the proposed Quantum Inspired Priority Scheduler (QIPS), a two-stage analysis was performed. First, we analyzed the QPSO component by varying particle size and iteration count to determine a stable, efficient configuration. This ensured that the optimization process operated under suitable parameter settings. Subsequently, the selected optimal configuration was compared with standard YARN schedulers, namely FIFO, Fair, and Capacity, to assess improvements in scheduling performance. The evaluation is based on key metrics, including latency, deadline adherence, resource utilization, data locality, and throughput.

3.1. QPSO Parameter Analysis

To ensure effective optimization, the QPSO component was evaluated under different parameter settings. The number of particles and iterations was varied to study their impact on scheduling performance. Experiments were conducted with particle sizes ranging from 10 to 30, and iteration counts from 50 to 100. The chosen range balances solution quality and computational efficiency, as lower iteration values led to unstable results, while higher values showed minimal improvement with increased overhead. The contraction expansion parameter (β) was kept constant, as preliminary observations indicated little influence on overall performance. Table 1 presents the corresponding results. The evaluated metrics include: P1 Makespan (s), P2 Average Latency (s), P3 Deadline Penalty (s), P4 Deadlines Met (%), P5 Resource Utilization, P6 Data Locality (%), P7 Priority Score, and P8 Throughput.

The experimental results indicate that the proposed QIPS scheduler effectively optimizes multiple objectives across different parameter settings. A moderate particle size ($n = 20$) provides balanced performance across key metrics, avoiding the variability seen with smaller swarms ($n = 10$) and the limited improvement with larger swarms ($n = 30$). In addition, the algorithm converges stably within 70–90 iterations, beyond which further increases yield only marginal performance gains. These observations

highlight QIPS's ability to balance exploration and exploitation, improving scheduling efficiency in Hadoop YARN environments. Based on this analysis, we select an optimal parameter configuration, as shown in Table 2, for subsequent comparison.

Table 1. QPSO Parameter Sensitivity Analysis

Iterations	Particles	P1	P2	P3	P4	P5	P6	P7	P8
10	50	678.11	242.5	1348	37.5	16.1	62.5	0.744	1.18
	60	705.53	230.25	1138	37.5	16.1	62.5	0.781	1.13
	70	703.7	196.25	947	50	16.1	62.5	0.91	1.14
	80	647.06	263.75	1474	50	16.1	75	0.9	1.24
	90	647.06	263.75	1474	50	16.1	75	0.9	1.24
	100	693.23	188.75	847	50	16.1	62.5	0.789	1.15
20	50	691.43	216	930	25	16.1	75	0.694	1.16
	60	703.7	196.25	947	50	16.1	62.5	0.91	1.14
	70	684.77	206	873	37.5	16.1	75	0.755	1.17
	80	691.43	216	930	25	16.1	75	0.694	1.16
	90	684.77	206	873	37.5	16.1	75	0.755	1.17
	100	703.7	196.25	947	50	16.1	62.5	0.91	1.14
30	50	704.45	193.88	928	50	16.1	62.5	0.862	1.14
	60	703.7	195.75	945	50	16.1	62.5	0.91	1.14
	70	703.7	195.75	945	50	16.1	62.5	0.91	1.14
	80	704.45	193.88	928	50	16.1	62.5	0.862	1.14
	90	704.45	193.88	928	50	16.1	62.5	0.862	1.14
	100	703.7	196.25	947	50	16.1	62.5	0.91	1.14

The QIPS scheduler demonstrates stable, balanced performance when configured with 20 particles and 70 iterations. This configuration achieves a desirable trade-off across multiple objectives, including makespan, latency, deadline satisfaction, data locality, and throughput. Therefore, it is selected as the representative configuration for comparison with existing schedulers. Furthermore, under certain parameter settings, such as 10 particles and 80 iterations, QIPS can achieve even lower makespan and higher throughput. However, this configuration exhibits higher latency and reduced stability across other metrics. Hence, while such configurations highlight the adaptability and optimization capability of QIPS, the balanced configuration (20, 70) is preferred for fair and consistent comparison.

Table 2. Optimal QPSO Configuration

Parameter	Value	Parameter	Value
Particles	20	Deadlines Met	37.5%
Iterations	70	Resource Utilization	16.1%
Makespan	684.77 s	Data Locality	75%
Average Latency	206 s	Priority Score	0.755
Deadline Penalty	873 s	Throughput	1.17

The convergence behavior of QPSO indicates how quickly the optimization process stabilizes as iterations progress. Faster convergence indicates efficient exploration of the solution space and reduced computational overhead. As shown in Fig. 4, the fitness value stabilizes within the initial iterations, indicating rapid convergence. This suggests that the algorithm reaches a near-optimal solution early, and additional iterations do not significantly improve performance.

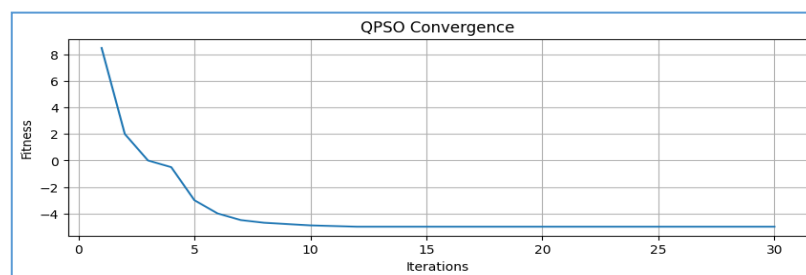


Fig. 4. Convergence behavior of QPSO for the selected configuration

3.2. Scheduler Comparison

Using the selected QPSO parameter settings, the performance of QIPS was compared with standard YARN schedulers (FIFO, Fair, and Capacity) and other previously discussed scheduling approaches. All schedulers were evaluated under the same workload and cluster conditions to ensure a fair comparison. This evaluation aims to examine how effectively each scheduler handles task execution, resource allocation, and priority management. The comparative results are presented in Table 3.

Table 3. Performance Comparison of QIPS with Baseline Schedulers

Scheduler	Makespan (s)	Avg Latency (s)	Deadline Penalty (s)	Deadlines Met (%)	Resource Usage (%)	Data Locality (%)	Priority Score	Throughput
Capacity	827.25	259.25	1645	25	16.1	0	0.622	0.97
Fair	700.13	284.25	1808	12.5	16.1	50	0.53	1.14
FIFO	662.93	264.38	1579	37.5	16.1	75	0.771	1.21
Frugal	724.23	255.5	1467	37.5	16.1	37.5	0.825	1.1
HFSP	827.25	271.25	1741	25	16.1	0	0.622	0.97
HybSMRP	719.7	212.62	1166	50	16.1	50	0.856	1.11
Intra-task	738.42	228.5	1376	37.5	16.1	25	0.799	1.08
QIPS	684.77	206	873	37.5	16.1	75	0.755	1.17

The comparative results highlight the proposed QIPS scheduler's effectiveness across multiple performance metrics. In terms of makespan, QIPS achieves a value of 684.77 s, which is significantly lower than most schedulers, such as Capacity (827.25 s), HFSP (827.25 s), Intra-task (738.42 s), and Frugal (724.23 s), and is only slightly higher than FIFO (662.93 s). This indicates that QIPS maintains competitive execution efficiency while avoiding the limitations of purely throughput-driven schedulers, such as FIFO. Regarding average latency, QIPS records the lowest value (206 s) among all schedulers, outperforming even HybSMRP (212.62 s), demonstrating its ability to reduce waiting time and improve responsiveness. This is particularly important for priority-driven, time-sensitive workloads. Fig. 5 presents a comparison of scheduler performance in terms of makespan and average latency.

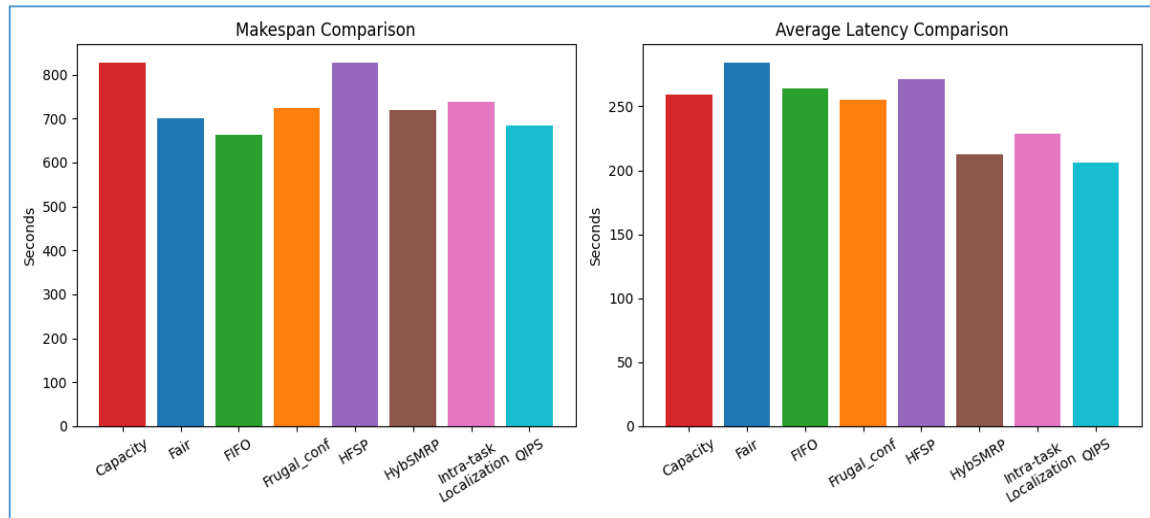


Fig. 5. Performance Comparison of Schedulers in Terms of Makespan and Latency

A key strength of QIPS is observed in the deadline penalty, where it achieves the lowest value (873), significantly outperforming all other schedulers. For instance, HybSMRP records a penalty of 1166, while FIFO and Fair exhibit much higher penalties of 1579 and 1808, respectively. This clearly indicates that QIPS is more effective in handling deadline-constrained tasks. In terms of deadlines met, QIPS achieves 37.5%, which, while not the highest (HybSMRP achieves 50%), still represents a balanced outcome given its superior latency and penalty performance. This reflects its multi-objective optimization capability rather than focusing on a single metric. Resource utilization remains nearly the same across all schedulers due to the fixed workload and cluster capacity. Since schedulers mainly affect

task execution order rather than total resource demand, this metric does not significantly differentiate their performance in this setup. Fig. 6 compares schedulers based on deadline penalty and the percentage of deadlines met.

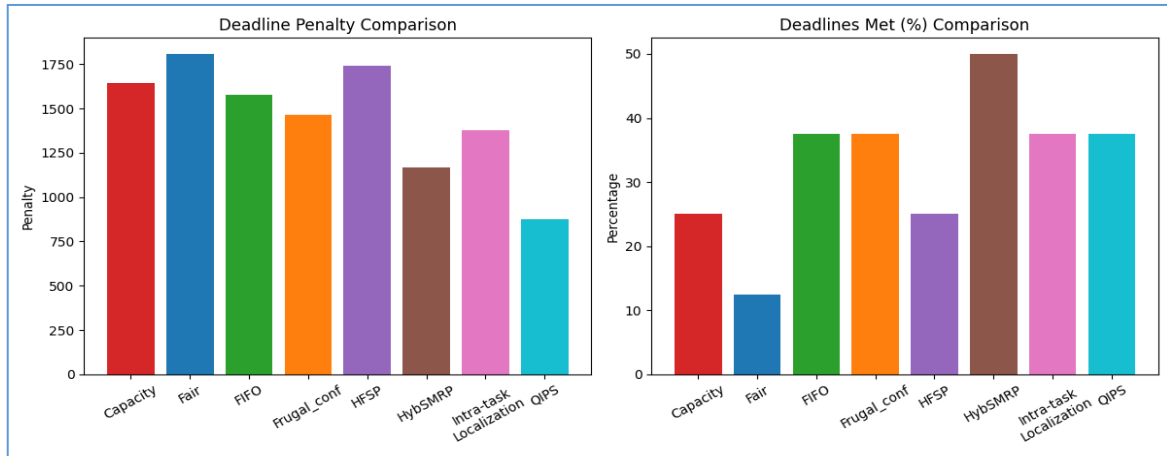


Fig. 6. Comparison of Deadline Penalty and Deadlines Met Across Schedulers

For data locality, QIPS achieves the highest level (75%), matching FIFO and outperforming all other schedulers. This reduces data transfer overhead and improves execution efficiency. Regarding priority score, QIPS (0.755) performs competitively, surpassing schedulers such as FIFO (0.771, slightly higher but less balanced), Fair (0.53), and Capacity (0.622), while remaining close to HybSMRP (0.856). This indicates effective handling of priority-aware scheduling without compromising other objectives. In terms of throughput, QIPS achieves 1.17, which is among the highest, second only to FIFO (1.21). Unlike FIFO, which prioritizes throughput at the cost of deadline performance, QIPS maintains high throughput while also optimizing latency and deadline-related metrics. Fig. 7 shows the comparison of schedulers in terms of data locality, priority score, and throughput.

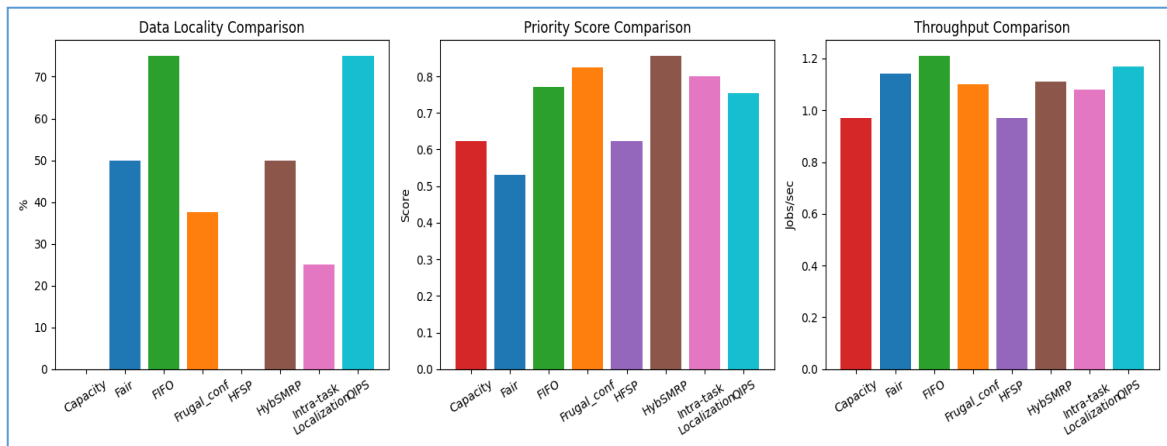


Fig. 7. Comparison of Data Locality, Priority Score, and Throughput Across Schedulers

Overall, the results indicate that while certain schedulers perform better in individual metrics, QIPS consistently achieves strong performance across all evaluated criteria. In particular, it demonstrates lower latency and deadline penalty while maintaining competitive makespan, throughput, and data locality. This highlights its ability to balance multiple objectives effectively, making it well-suited for priority-aware and deadline-sensitive scheduling in Hadoop YARN environments.

4. Conclusion

This study presented a Quantum-Inspired Priority Scheduler (QIPS) that integrates QPSO-based optimization with priority-aware and locality driven scheduling in the Hadoop YARN framework. The experimental results demonstrate that QIPS improves scheduling performance compared to conventional

Hadoop schedulers, particularly by reducing deadline penalties and latency while maintaining competitive makespan, throughput, and data locality. These results highlight the effectiveness of quantum inspired optimization in addressing multi objective scheduling challenges in large-scale distributed systems. Overall, QIPS provides a balanced and adaptive scheduling approach and is readily extendable to larger cluster environments. Despite these promising results, several directions remain for future work. QIPS can be extended to support containerized and cloud-native platforms, including integration with Kubernetes based systems. Incorporating advanced learning techniques, such as reinforcement learning or neuroevolutionary methods, may further enhance adaptability to dynamic workloads. Additionally, evaluating QIPS on larger clusters and real-world datasets would improve its practical relevance. Exploring energy-aware and carbon-efficient scheduling strategies also presents an important opportunity toward building more sustainable distributed computing systems.

Acknowledgment

B. P and S. D. R. would like to thank Mr. Rohith S. V. (Cyber Trinetra) for his assistance in dashboard development and deployment support.

Declarations

Author contribution. Bhavana P.: Conceptualization, methodology, investigation, data curation, writing original draft preparation. Shashikumar D. R.: Supervision, review, and editing. All authors had approved the final version.

Funding statement. This research received no external funding.

Conflict of interest. The authors declare no conflict of interest.

Additional information. No additional information is available for this paper.

References

- [1] S. Hedayati, N. Maleki, T. Olsson, F. Ahlgren, M. Seyednezhad, and K. Berahmand, "MapReduce scheduling algorithms in Hadoop: a systematic study," *J. Cloud Comput.*, vol. 12, no. 1, p. 143, 2023, doi: [10.1186/s13677-023-00520-9](https://doi.org/10.1186/s13677-023-00520-9).
- [2] M. Hussain, L.-F. Wei, A. Rehman, M. Ali, S. M. Waqas, and F. Abbas, "Cost-aware quantum-inspired genetic algorithm for workflow scheduling in hybrid clouds," *J. Parallel Distrib. Comput.*, vol. 191, p. 104920, 2024, doi: [10.1016/j.jpdc.2024.104920](https://doi.org/10.1016/j.jpdc.2024.104920).
- [3] S. K. Sood, "Quantum-inspired metaheuristic algorithms for Industry 4.0: A scientometric analysis," *Eng. Appl. Artif. Intell.*, vol. 139, p. 109635, 2025, doi: [10.1016/j.engappai.2024.109635](https://doi.org/10.1016/j.engappai.2024.109635).
- [4] R. Ghazali, S. Adabi, A. Rezaee, D. G. Down, and A. Movaghar, "CLQLMRS: improving cache locality in MapReduce job scheduling using Q-learning," *J. Cloud Comput.*, vol. 11, no. 1, p. 45, 2022, doi: [10.1186/s13677-022-00322-5](https://doi.org/10.1186/s13677-022-00322-5).
- [5] A. M. Rahmani, E. Y. Chamzini, M. Pourshaban, and M. Hosseinzadeh, "Scheduling of big data workflows in the Hadoop framework with heterogeneous computing cluster," *Arab. J. Sci. Eng.*, vol. 50, no. 15, pp. 12449–12461, 2025, doi: [10.1007/s13369-024-09779-9](https://doi.org/10.1007/s13369-024-09779-9).
- [6] Z. Zhang, C. Xu, S. Xu, L. Huang, and J. Zhang, "Towards optimized scheduling and allocation of heterogeneous resource via graph-enhanced EPSO algorithm," *J. Cloud Comput.*, vol. 13, no. 1, p. 108, 2024, doi: [10.1186/s13677-024-00670-4](https://doi.org/10.1186/s13677-024-00670-4).
- [7] B. Qureshi, "Optimizing Hadoop Scheduling in Single-Board-Computer-Based Heterogeneous Clusters," *Computation*, vol. 12, no. 5, p. 96, 2024, doi: [10.3390/computation12050096](https://doi.org/10.3390/computation12050096).
- [8] U. K. Lilhore *et al.*, "QHRMOF: A Quantum-Inspired hybrid Multi-Objective framework for Energy-Efficient task scheduling and load balancing in cloud computing," *J. Cloud Comput.*, vol. 14, no. 1, p. 54, 2025, doi: [10.1186/s13677-025-00777-2](https://doi.org/10.1186/s13677-025-00777-2).
- [9] M. Cui and Y. Wang, "An effective QoS-aware hybrid optimization approach for workflow scheduling in cloud computing," *Sensors*, vol. 25, no. 15, p. 4705, 2025, doi: [10.3390/s25154705](https://doi.org/10.3390/s25154705).
- [10] K. Fu, J. Liu, M. Chen, and H. Zhang, "Solving flexible job-shop scheduling problems based on quantum computing," *Entropy*, vol. 27, no. 2, p. 189, 2025, doi: [10.3390/e27020189](https://doi.org/10.3390/e27020189).

- [11] S. P. Singh, G. Kumar, U. Ahirwar, S. Selvarajan, and F. Khan, "Multi-objective quantum hybrid evolutionary algorithms for enhancing quality-of-service in internet of things," *Sci. Rep.*, vol. 15, no. 1, p. 14795, 2025, doi: [10.1038/s41598-025-99429-3](https://doi.org/10.1038/s41598-025-99429-3).
- [12] Y. Bai, Y. Sui, X. Deng, and X. Wang, "Quantum-inspired robust optimization for coordinated scheduling of PV-hydrogen microgrids under multi-dimensional uncertainties," *Sci. Rep.*, vol. 15, no. 1, p. 29589, 2025, doi: [10.1038/s41598-025-12280-4](https://doi.org/10.1038/s41598-025-12280-4).
- [13] H. Tang and J. Dong, "Solving flexible job-shop scheduling problem with heterogeneous graph neural network based on relation and deep reinforcement learning," *Machines*, vol. 12, no. 8, p. 584, 2024, doi: [10.3390/machines12080584](https://doi.org/10.3390/machines12080584).
- [14] Z. Du, J. Wang, and H. Li, "Quantum firefly algorithm: a novel approach for quantum circuit scheduling optimization," *Electronics*, vol. 14, no. 11, p. 2123, 2025, doi: [10.3390/electronics14112123](https://doi.org/10.3390/electronics14112123).
- [15] D. Fitzek, T. Ghandriz, L. Laine, M. Granath, and A. F. Kockum, "Applying quantum approximate optimization to the heterogeneous vehicle routing problem," *Sci. Rep.*, vol. 14, no. 1, p. 25415, 2024, doi: [10.1038/s41598-024-76967-w](https://doi.org/10.1038/s41598-024-76967-w).
- [16] F. Shabestari, A. M. Rahmani, N. J. Navimipour, and S. Jabbehdari, "A YARN-based Energy-Aware Scheduling Method for Big Data Applications under Deadline Constraints: F. Shabestari et al.," *J. Grid Comput.*, vol. 20, no. 4, p. 38, 2022, doi: [10.1007/s10723-022-09627-w](https://doi.org/10.1007/s10723-022-09627-w).
- [17] M. Bergui, S. Hourri, S. Najah, and N. S. Nikolov, "Predictive modelling of MapReduce job performance in cloud environments using machine learning techniques," *J. Big Data*, vol. 11, no. 1, p. 98, 2024, doi: [10.1186/s40537-024-00964-z](https://doi.org/10.1186/s40537-024-00964-z).
- [18] B. Qureshi, "Adaptive Multi-Criteria Selection for Efficient Resource Allocation in Frugal Heterogeneous Hadoop Clusters," *Electronics*, vol. 13, no. 10, p. 1836, 2024, doi: [10.3390/electronics13101836](https://doi.org/10.3390/electronics13101836).
- [19] Y. Li, W. Song, B. Jin, X. Zuo, Y. Li, and K. Chen, "A SqueeSAR Spatially Adaptive Filtering Algorithm Based on Hadoop Distributed Cluster Environment," *Appl. Sci.*, vol. 13, no. 3, p. 1869, 2023, doi: [10.3390/app13031869](https://doi.org/10.3390/app13031869).
- [20] J. Zhou, B. Liu, and J. Gao, "A task scheduling algorithm with deadline constraints for distributed clouds in smart cities," *PeerJ Comput. Sci.*, vol. 9, p. e1346, 2023, doi: [10.7717/peerj-cs.1346](https://doi.org/10.7717/peerj-cs.1346).
- [21] J. Wang, S. Li, X. Zhang, F. Wu, and C. Xie, "Deep reinforcement learning task scheduling method based on server real-time performance," *PeerJ Comput. Sci.*, vol. 10, p. e2120, 2024, doi: [10.7717/peerj-cs.2120](https://doi.org/10.7717/peerj-cs.2120).
- [22] Y. Sang, J. Cheng, B. Wang, and M. Chen, "A three-stage heuristic task scheduling for optimizing the service level agreement satisfaction in device-edge-cloud cooperative computing," *PeerJ Comput. Sci.*, vol. 8, p. e851, 2022, doi: [10.7717/peerj-cs.851](https://doi.org/10.7717/peerj-cs.851).
- [23] D. Alsadie, "Advancements in heuristic task scheduling for IoT applications in fog-cloud computing: challenges and prospects," *PeerJ Comput. Sci.*, vol. 10, p. e2128, 2024, doi: [10.7717/peerj-cs.2128](https://doi.org/10.7717/peerj-cs.2128).
- [24] P. Thanapol, K. Lavangnananda, F. Leprévost, A. Glad, J. Schleich, and P. Bouvry, "Round-based mechanism and job packing with model-similarity-based policy for Scheduling DL Training in GPU Cluster," *Appl. Sci.*, vol. 14, no. 6, p. 2349, 2024, doi: [10.3390/app14062349](https://doi.org/10.3390/app14062349).
- [25] V. Thesma, G. C. Rains, and J. Mohammadpour Velni, "Development of a low-cost distributed computing pipeline for high-throughput cotton phenotyping," *Sensors*, vol. 24, no. 3, p. 970, 2024, doi: [10.3390/s24030970](https://doi.org/10.3390/s24030970).
- [26] S. N. Hegde, D. B. Srinivas, M. A. Rajan, S. Rani, A. Kataria, and H. Min, "Multi-objective and multi constrained task scheduling framework for computational grids," *Sci. Rep.*, vol. 14, no. 1, p. 6521, 2024, doi: [10.1038/s41598-024-56957-8](https://doi.org/10.1038/s41598-024-56957-8).
- [27] V. P. Verma, S. Kumar, S. Kumar, N. S. Naik, and R. Dubey, "Optimizing Spark job scheduling with distributional deep learning in cloud environments," *J. Cloud Comput.*, vol. 14, no. 1, p. 59, 2025, doi: [10.1186/s13677-025-00773-6](https://doi.org/10.1186/s13677-025-00773-6).
- [28] A. C. Ikegwu, H. F. Nweke, E. Mkpojiogu, C. V. Anikwe, S. A. Igwe, and U. R. Alo, "Recently emerging trends in big data analytic methods for modeling and combating climate change effects," *Energy Informatics*, vol. 7, no. 1, pp. 1–28, 2024, doi: [10.1186/s42162-024-00307-5](https://doi.org/10.1186/s42162-024-00307-5).

- [29] S. Vengadeswaran, S. R. Balasundaram, and P. Dhavakumar, "IDaPS Improved data-locality aware data placement strategy based on Markov clustering to enhance MapReduce performance on Hadoop," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 36, no. 3, p. 101973, 2024, doi: [10.1016/j.jksuci.2024.101973](https://doi.org/10.1016/j.jksuci.2024.101973).
- [30] M. Thiyyakat, S. Kalambur, and D. Sitaram, "Constraint-Aware Federated Scheduling for Data Center Workloads," *IoT*, vol. 4, no. 4, pp. 534–557, 2023, doi: [10.3390/iot4040023](https://doi.org/10.3390/iot4040023).
- [31] N. Du, C. Wu, A. Hou, W. Nie, and R. Song, "Efficient Scheduling for GPU-Based Neural Network Training via Hybrid Reinforcement Learning and Metaheuristic Optimization," *Big Data Cogn. Comput.*, vol. 9, no. 11, p. 284, 2025, doi: [10.3390/bdcc9110284](https://doi.org/10.3390/bdcc9110284).
- [32] H. Wang, E. Deng, J. Li, and C. Zhang, "Edge computing resource scheduling method based on container elastic scaling," *PeerJ Comput. Sci.*, vol. 10, p. e2379, 2024, doi: [10.7717/peerj-cs.2379](https://doi.org/10.7717/peerj-cs.2379).
- [33] Y. Teng and Z. Liu, "Edge computing task scheduling method based on user's social relations: a construction and solution for Smart City Library," *PeerJ Comput. Sci.*, vol. 10, p. e2457, 2024, doi: [10.7717/peerj-cs.2457](https://doi.org/10.7717/peerj-cs.2457).
- [34] A. Muhammad and M. A. Qadir, "MF-Storm: a maximum flow-based job scheduler for stream processing engines on computational clusters to increase throughput," *PeerJ Comput. Sci.*, vol. 8, p. e1077, 2022, doi: [10.7717/peerj-cs.1077](https://doi.org/10.7717/peerj-cs.1077).
- [35] M. Elhmadany, I. Elmadah, and H. E. Abdelmunim, "Instance segmentation on distributed deep learning big data cluster," *J. Big Data*, vol. 11, no. 1, p. 6, 2024, doi: [10.1186/s40537-023-00871-9](https://doi.org/10.1186/s40537-023-00871-9).
- [36] L. Chourasiya *et al.*, "Advanced system log analyzer for anomaly detection and cyber forensic investigations using LSTM and transformer networks," *J. Cloud Comput.*, vol. 14, no. 1, p. 60, 2025, doi: [10.1186/s13677-025-00789-y](https://doi.org/10.1186/s13677-025-00789-y).
- [37] Z. Wu, X. Lv, Y. Yun, and W. Duan, "A parallel sequential SBAS processing framework based on Hadoop distributed computing," *Remote Sens.*, vol. 16, no. 3, p. 466, 2024, doi: [10.3390/rs16030466](https://doi.org/10.3390/rs16030466).
- [38] D. Rahbari, "Analyzing meta-heuristic algorithms for task scheduling in a fog-based IoT application," *Algorithms*, vol. 15, no. 11, p. 397, 2022, doi: [10.3390/a15110397](https://doi.org/10.3390/a15110397).
- [39] M. N. Aktan and H. Bulut, "Metaheuristic task scheduling algorithms for cloud computing environments," *Concurr. Comput. Pract. Exp.*, vol. 34, no. 9, p. e6513, 2022, doi: [10.1002/cpe.6513](https://doi.org/10.1002/cpe.6513).
- [40] C. Wang, Z. Wang, S. Zhang, X. Liu, and J. Tan, "Reinforced quantum-behaved particle swarm-optimized neural network for cross-sectional distortion prediction of novel variable-diameter-die-formed metal bent tubes," *J. Comput. Des. Eng.*, vol. 10, no. 3, pp. 1060–1079, 2023, doi: [10.1093/jcde/qwad037](https://doi.org/10.1093/jcde/qwad037).
- [41] B. Liu, Y. Zhou, Q. Luo, and H. Huang, "Quantum-inspired African vultures optimization algorithm with elite mutation strategy for production scheduling problems," *J. Comput. Des. Eng.*, vol. 10, no. 4, pp. 1767–1789, 2023, doi: [10.1093/jcde/qwad078](https://doi.org/10.1093/jcde/qwad078).
- [42] A. Dong and S.-K. Lee, "The study of an improved particle swarm optimization algorithm applied to economic dispatch in microgrids," *Electronics*, vol. 13, no. 20, p. 4086, 2024, doi: [10.3390/electronics13204086](https://doi.org/10.3390/electronics13204086).
- [43] S. Lim and D. Park, "Improving Hadoop MapReduce performance on heterogeneous single board computer clusters," *Futur. Gener. Comput. Syst.*, vol. 160, pp. 752–766, 2024, doi: [10.1016/j.future.2024.06.025](https://doi.org/10.1016/j.future.2024.06.025).
- [44] H. Hou, S. N. A. Jawaddi, and A. Ismail, "Energy efficient task scheduling based on deep reinforcement learning in cloud environment: A specialized review," *Futur. Gener. Comput. Syst.*, vol. 151, pp. 214–231, 2024, doi: [10.1016/j.future.2023.10.002](https://doi.org/10.1016/j.future.2023.10.002).
- [45] M. Hosseinzadeh, M. Y. Ghafour, H. K. Hama, B. Vo, and A. Khoshnevis, "Multi-Objective Task and Workflow Scheduling Approaches in Cloud Computing: a Comprehensive Review: Multi-objective Task and Workflow Scheduling Approaches in Cloud Computing: A Comprehensive Review," *J. Grid Comput.*, vol. 18, no. 3, pp. 327–356, 2020, doi: [10.1007/s10723-020-09533-z](https://doi.org/10.1007/s10723-020-09533-z).
- [46] S. A. Murad, A. J. M. Muzahid, Z. R. M. Azmi, M. I. Hoque, and M. Kowsher, "A review on job scheduling technique in cloud computing and priority rule based intelligent framework," *J. King Saud Univ. Inf. Sci.*, vol. 34, no. 6, pp. 2309–2331, 2022, doi: [10.1016/j.jksuci.2022.03.027](https://doi.org/10.1016/j.jksuci.2022.03.027).
- [47] G. Kotikam and L. Selvaraj, "YARN Schedulers for Hadoop MapReduce Jobs: Design Goals, Issues and Taxonomy," *Recent Adv. Comput. Sci. Commun. (Formerly Recent Patents Comput. Sci.)*, vol. 16, no. 6, pp. 44–55, 2023, doi: [10.2174/2666255816666220831125012](https://doi.org/10.2174/2666255816666220831125012).