Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.075063

ARTICLE

A Workflow Scheduling Method Based on the Combination of Tunicate Swarm Algorithm and Highest Response Ratio Next Scheduling

Yujie Tian[1], Ming Zhu[1], Jing Li[1,*], Cong Liu[2] and Ziyang Zhang[1]

[1]School of Computer Science and Technology, Shandong University of Technology, Zibo, China
[2]NOVA Information Management School, Universidade Nova de Lisboa, Campus de Campolide, Lisboa, Portugal

*Corresponding Author: Jing Li. Email: li_jing@sdut.edu.cn

Received: 24 October 2025; Accepted: 16 January 2026; Published: 12 March 2026

ABSTRACT: Workflow scheduling is critical for efficient cloud resource management. This paper proposes Tunicate Swarm-Highest Response Ratio Next, a novel scheduler that synergistically combines the Tunicate Swarm Algorithm with the Highest Response Ratio Next policy. The Tunicate Swarm Algorithm generates a cost-minimizing task-to-VM mapping scheme, while the Highest Response Ratio Next dynamically dispatches tasks in the ready queue with the highest-priority. Experimental results demonstrate that the Tunicate Swarm-Highest Response Ratio Next reduces costs by up to 94.8% compared to meta-heuristic baselines. It also achieves competitive cost efficiency vs. a learning-based method while offering superior operational simplicity and efficiency, establishing it as a highly practical solution for dynamic cloud environments.

KEYWORDS: Workflow scheduling; cloud computing; tunicate swarm algorithm; highest response ratio next scheduling

1 Introduction

The cloud environment offers a wide range of diverse computing resources along with reliable data storage options, so many institutions are increasingly using cloud computing to handle workflows [1]. In the scheduling process, it is essential to rent and allocate suitable Virtual Machines (VMs) to execute a series of tasks or requests while minimizing both execution time and expenses [2]. Within cloud computing, workflow scheduling focuses on improving task deployment speed and lowering infrastructure costs by dynamically allocating tasks to appropriate virtual machines.

This type of problem is NP complete [3]. Existing scheduling approaches can be broadly categorized into several strands, each with distinct limitations. Traditional methods [4], such as list scheduling, are computationally efficient but often fail to address multi objective optimization under deadline constraints or adapt to dynamic, heterogeneous cloud environments. Heuristic and meta-heuristic algorithms (e.g., Genetic Algorithm (GA), Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO)) have been widely applied to workflow scheduling [5]. However, they frequently suffer from complex parameter tuning, high computational complexity as the problem scale increases, and slow convergence, making it challenging to minimize operational costs and avoid SLA (Service Level Agreement) violations efficiently [6–8]. We identify that their coupled decision-making process entangling strategic planning with operational dispatch is a primary source of these inefficiencies, leading to inherent trade offs between cost minimization and responsiveness.

More recent approaches seek greater adaptability. For instance, hyper-heuristic methods operate at a higher level of abstraction, aiming to select or dynamically generate heuristics [1]. While this promises flexibility, their performance is often contingent on the quality and diversity of the underlying heuristic pool, and significant overhead incurs in evaluating and switching between strategies. Similarly, Reinforcement Learning (RL)-based schedulers learn optimal policies through interaction with the environment [9,10] but often require extensive training data, sophisticated reward function, and may lack stability in non-stationary cloud settings. Both hyper-heuristic and the RL approaches hinder interpretability and real-time decision-making in production systems.

To address these challenges, this paper proposes the Tunicate Swarm-High Response Ratio Next (TS-HRRN) method, which employs a static two-layer architecture with highly responsive to dynamic workloads. It synergistically couples the Tunicate Swarm Algorithm (TSA) for global, cost-aware VM allocation with the HRRN policy for local, responsiveness-driven task dispatch. This fixed assignment of distinct, complementary roles eliminates the runtime overhead of dynamic algorithm selection or model inference, ensuring inherent stability and efficiency. Extensive evaluations against state-of-the-art schedulers demonstrate the significant improvements in cost-effectiveness and performance achieved by the TS-HRRN.

In summary, this paper makes the following main contributions:

- We propose the TS-HRRN method for the heterogeneous workflow scheduling problem, which aims to minimize total rental cost while preventing SLA violations.
- In the proposed TS-HRRN method, the TSA is used to select the VMs with the highest priority, and the HRRN is used to prioritize the tasks to be executed. The proposed TS-HRRN method shows potential to improve scientific workflow scheduling strategies.
- We use extensive experiments on real scientific workflow datasets to verify the effectiveness and efficiency of the proposed algorithm. Experimental results show that the TS-HRRN is superior to the referring algorithms in terms of cost minimization and algorithm stability.

The structure of the article is as follows: Section 2 introduces background knowledge; Section 3 introduces relevant work; Section 4 gives the collaborative use between two algorithms; Experimental and analytical results are given in Section 5; Finally, Section 6 summarizes the article.

2 Background Knowledge

This section introduces the background knowledge and definitions.

2.1 Workflow

In cloud computing, a workflow is modeled as a Directed Acyclic Graph (DAG) in which each node represents an independent computational task and each edge specifies data or control dependency between tasks. Following the established cost model and definitions of cloud workflow scheduling [1], we formalize the problem as follows.

Definition 1: *The ith workflow is defined as:* $W_i = (DAG_i, AT_i, NOR_i, DL_i, RDL_i)$.

Among them, i varies from 1 to m, DAG_i *is a workflow mode,* AT_i *indicates the arrival time, whereas* NOR_i *refers to the quantity of tasks that have not yet been assigned within the workflow,* DL_i *represents the deadline, once the workflow execution violates the deadline, corresponding compensation liability is triggered.* RDL_i *represents the remaining time.*

The remaining time is calculated as follows:

$$RDL_i = DL_i - current\ time \tag{1}$$

The workflow in the ready queue is handled by a collection of VMs ($V = \{V_1, V_2, \ldots, V_n\}$). Each VM instance is characterized as:

$V_k = (\ TYPE_k, NIQ_k, TIQ_k, VMR_k)$.

where k varies from 1 to n, $TYPE_k = (CU_k, MEM_k, PRICE_k)$ represents its VM type. CU_k is the unit of calculation (i.e., computing power), MEM_k represents the memory capacity, while $PRICE_k$ indicates the hourly rental fee imposed by the cloud provider. NIQ_k refers to the number of tasks in the queue of V_k, TIQ_k denotes the cumulative execution time for tasks in the queue, VMR_k signifies the remaining lease duration for V_k. All of these definitions are specific to the scheduling model adopted in this paper.

We aim to lower the total expenses of workflow execution, including VM rental and penalties:

$$TotalCost = \sum_{k \in LVMS} RentFee_k + \sum_{i \in W} Penalty_i \tag{2}$$

$$s.t. RentFee_k = PRICE_k \times \left\lceil \frac{FT_t - ST_t}{3600} \right\rceil \tag{3}$$

$$Penalty_i = \delta \times \max\{0, AT_i + Makespan_i - Deadline_i\} \tag{4}$$

$$Deadline_i = AT_i + \varepsilon \times Makespan_i \tag{5}$$

In the formulas above, W (resp. $LVMS$) is a collection of workflows (resp. rented VM instances). $RentFee_k$ represents the rental fee of V_k, $Penalty_i$ represents the penalty for violating the deadline of W_i. FT_t (resp. ST_t) represents the completion (resp. start) time of the final task on V_k.

Definition 2: *RentFee: an hourly cost for the VM offered by the worldwide cloud service industry [1,11,12]. The denominator and upper limit in Eq. (3) collectively transform this time interval into the total rental duration.*

AT_i signifies the arrival time of W_i, ε is relaxation coefficient [6]. $Makespan_i$ denotes the optimal earliest completion time of W_i. If $Deadline_i$ is violated as a result of a holdup in processing W_i, Eq. (4) is imposed, where δ represents the penalty factor. In our experiments, we set $\delta =$ 0.24/h, aligning with the unit penalty rate adopted in a prior cloud scheduling study [1] to ensure a comparable cost basis. A lower value indicates a higher tolerance for exceeding the deadline.

2.2 The Highest Response Ratio Next Scheduling

The workflow is represented as a DAG, where each node corresponds to an independent task and each edge indicates dependency—a predecessor task must be completed before its successor executes. The system first identifies schedulable tasks, defined as nodes whose predecessors have executed. These tasks constitute the Ready Queue. To determine an appropriate execution order, the HRRN algorithm is employed, which computes a response ratio for each task based on its waiting time and estimated service time, thereby favoring tasks that contribute to the overall system responsiveness.

The response ratio [13] R is calculated as:

$$R = 1 + WaitT/S \tag{6}$$

where $WaitT$ is the waiting time, and S is the required service time. The HRRN selects the task with the highest response ratio.

2.3 The Mathematical Model of the TSA

The TSA is a meta-heuristic that emulates the jet propulsion and swarm behaviors of tunicates during foraging. Its mathematical model comprises two core processes.

2.3.1 Jet Propulsion Process of Tunicates

Tunicate achieves vertical migration behavior through its jet propulsion, which also includes the process of moving from one position in a specific direction to another.

a. To avoid conflicts between search individuals: a positional variable A is introduced to calculate the new position of each searched individual, effectively avoiding collision problems during searching.

$$A = \frac{G}{M} \tag{7}$$

$$G = c_2 + c_3 - F \tag{8}$$

$$F = 2 \cdot c_1 \tag{9}$$

$$M = \left\lfloor P_{min} + c_1 \cdot P_{max} - P_{min} \right\rfloor \tag{10}$$

Among them, G represents gravity, M is the interaction force between two search agents, with variables c_1, c_2 and c_3 representing random values within the range of [0–1], P_{min} and P_{max} depict the starting velocity along with the velocity influenced by the interaction force among individuals, respectively. Generally, these two velocities are set to 1 and 4, respectively, F is the advection of water in the deep sea.

b. Moving towards the best position: it is essential to calculate the location of the food source and the separation between searching entities to identify the optimal position. In the first iteration, the separation of the searched entity from the food source is:

$$PD = FS - r \cdot P(x) \tag{11}$$

PD represents the separation between the food source and the encapsulated entity, FS indicates the position of the food source, $P(x)$ indicates the position of encapsulated individuals of the x th iteration, r is a random number between [0–1].

c. Approaching the optimal position: the tunicate reaches a new position and updates according to the following formula:

$$P(x) = \begin{cases} FS + A \cdot PD, & r \geq 0.5 \\ FS - A \cdot PD, & r < 0.5 \end{cases} \tag{12}$$

2.3.2 The Swarm Behavior

After resolving individual conflicts and calculating positions with the food source, tunicate uses swarm intelligence to approach the optimal food source. We use Formula (13) to simulate the swarm intelligence of tunicate:

$$P_{final}(x+1) = \frac{P(x) + P_{candidate}(x+1)}{2 + c_1} \tag{13}$$

$P(x)$ represents the current position of a search agent of iteration x. $P_{candidate}(x+1)$ denotes the new candidate positions, $P_{final}(x+1)$ is the finalized position of the agent of $x+1$.

3 Related Work

3.1 Optimization Objective

Early workflow research primarily focused on minimizing execution time. Rimal et al. leveraged idle cloud resources to reduce workflow makespan [14]. Meanwhile, Ahmed et al. proposed an enhanced HEFT algorithm that improved task prioritization and resource mapping [15]. Subsequently, scheduling objectives expanded to incorporate cost and deadline requirements. Djigal et al. introduced a two-stage list scheduling algorithm that reduced both cost and time [16]. Sun et al. optimized scheduling time by integrating critical path analysis and task merging into the HEFT algorithm [17]. In a similar vein, Chen et al. enhanced the Min-Min algorithm by treating data transmission time as a critical optimization factor [18].

3.2 Optimization Algorithm

Heuristic algorithms have been widely applied to workflow scheduling and have yielded substantial results. For instance, Mohammadzadeh et al. combined the Seagull and Grasshopper Optimization Algorithms to optimize cost and time; however, the complexity of their proposed approach may hinder its practical application [19]. Similarly, Mikram et al. integrated the PSO and GA to reduce completion time but did not account for dynamic environments [5]. Zeedan et al. employed an Enhanced Binary ABC within a Pareto frontier model, which demonstrated performance on specific problems [20]. Other notable approaches include Iranmanesh et al., who enhanced genetic operators for effective load balancing under deadline constraints [21], and Varshney et al., who leveraged an ant colony algorithm to optimize task migration for shorter time and lower cost [22]. Additionally, Chakravarthi et al. incorporated an encoding scheme to generate cost-effective schedules [23].

Beyond traditional heuristics, more advanced paradigms such as hyper-heuristic [1] and the RL [9,10] have been explored to enhance the adaptability. However, as discussed before, these methods have computational overhead and complexity in pursuit of generality, which motivates us to design a decoupled, yet responsive and cost-effective hybrid architecture.

3.3 Motivation for Algorithm Selection

Traditional workflow schedulers employed monolithic framework to simultaneously optimize task sequencing and resource allocation. This coupled approach forces a single algorithm to make conflicting decisions, leading to a fundamental trade-off: methods effective in global, cost-sensitive resource planning are often too slow to respond to dynamic task arrivals, while fast, responsive task dispatchers lack the global view necessary for overall cost minimization. Our TS-HRRN method breaks this paradigm by introducing a novel dual-priority scheduling architecture that strategically decouples these two concerns. In this architecture, the HRRN policy acts as a dedicated high frequency scheduler to dynamically prioritize tasks in the ready queue, thereby ensuring immediate system responsiveness. The TSA focuses on long term cost efficiency and the SLA compliance. The scheduler executes the optimal match by assigning the highest priority task from the HRRN queue to the most suitable VM as prescribed by the TSA generated scheme.

4 Process and Algorithm

4.1 Execution Process of the Method

This article proposes collaborative scheduling of dynamic arrival workflows using the HRRN and the TSA. Fig. 1 illustrates the execution process: The high-level workflow structure is adapted from prior work [1], the core technical components and their interactions are original and depict our novel TS-HRRN

architecture. When the workflow is uploaded to the cloud, the TSA is employed to generate an optimal task-to-VM matching scheme. Different tasks of the workflow are assigned to different VMs. The fitness function defines the overall cost and drives the global optimization. The HRRN is used to prioritize the tasks.

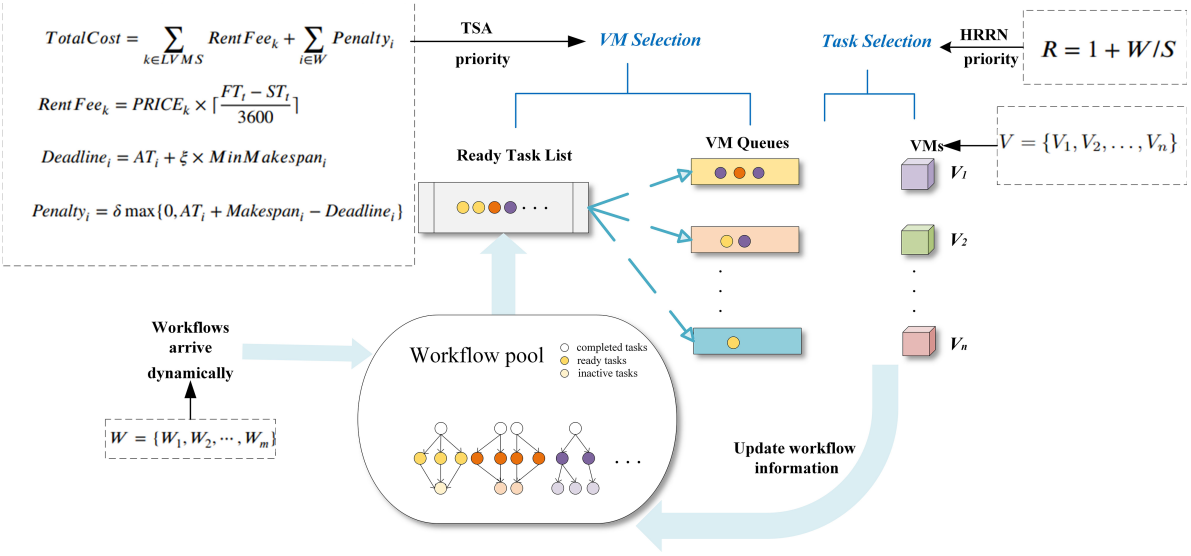


Figure 1: The execution process of the workflow.

Mapping the TSA to Cloud Workflow Scheduling

To apply the TSA to cloud workflow scheduling, each candidate solution (i.e., an individual of the population) represents a specific task-to-VM allocation scheme. By defining a fitness function that incorporates the execution time, the VM rental cost, and the SLA violation penalties, the TSA iteratively minimizes the total cost. The collective migration behavior of the swarm simulates the cost-driven optimization process, guiding the population toward more efficient scheduling solutions.

4.2 Algorithm Structure

Algorithm 1 presents the main scheduling loop of the TS-HRRN, which implements the dual-layer architecture through coordinated execution. The algorithm initializes the simulation parameters, including the time, queues, and a global task-to-VM mapping scheme (lines 1–6). The core loop operates in four phases. First, it adds new tasks to the ready queue (lines 10–18). Second, it invokes the TSA as a global optimizer to regenerate the mapping scheme (lines 21–23). Third, the HRRN dispatches the highest-priority task by assigning it to the VM in the current mapping scheme and updates execution timelines (lines 26–38). Finally, the simulation advances to the next event time (lines 41–42). This ensures global optimization through periodic the TSA execution while maintaining responsiveness via the HRRN dispatch.

This decoupled architecture, comprising two layers, inherently adjusts to dynamic workloads. The HRRN scheduler handles microscopic task queue changes (e.g., arrivals, resolved dependencies) in real time, guaranteeing low latency. In parallel, the TSA module operates as a periodic strategic planner, which re-optimizes the global task to the VM mapping by processing a current system snapshot of all ready tasks and available resources.

Algorithm 2 presents the TSA global optimization procedure. The algorithm initializes a population of candidate task-to-VM mappings and iteratively refines them by emulating tunicate swarm behavior. Solutions are evaluated by the fitness function (Eq. (2)). Algorithm 2 adjusts each individual's position based

on conflict avoidance, movement towards the best neighbor, and swarm intelligence. It returns the solution with the minimized cost while satisfying the SLA requirements.

Algorithm 3 defines the fitness assessment for complete scheduling solutions. It simulates task execution under a given mapping scheme while respecting task dependencies. The function processes tasks in topological order, calculates VM busy intervals, aggregates rental costs based on actual usage, and sums deadline violation penalties using predefined SLA parameters. The final fitness value represents the combined cost of resource consumption and SLA violations.

Algorithm 1: TS-HRRN main scheduling loop

Input: List of workflows W, List of VMs V
Output: Scheduled tasks with assigned VMs and timestamps

$currentTime \leftarrow 0$
$readyQueue, scheduledTasks, mappingScheme \leftarrow \varnothing$ ▷ Global TSA-generated mapping
$lastOptimizationTime \leftarrow 0$
$optimizationInterval \leftarrow \Delta t$ ▷ TSA runs every Δt time units
$taskFinishTimes \leftarrow \{\}$ ▷ Track task completion times for dependencies
while unscheduled tasks remain **do**
  Step 1: Add Ready Queue
  for each workflow $W_i \in W$ **do**
    if $W_i.arrivalTime \leq currentTime$ **then**
      for each task t in W_i **do**
        if t not scheduled and $t \notin readyQueue$ and predecessors of t are completed **then**
          $readyQueue \leftarrow readyQueue \bigcup t$
        end if
      end for
    end if
  end for
  Step 2: Periodic Global Optimization via TSA
  if $currentTime - lastOptimizationTime \geq optimizationInterval$ **then**
    $mappingScheme \leftarrow$ TSA_GlobalOptimize$(readyQueue, V)$
    $lastOptimizationTime \leftarrow currentTime$
  end if
  Step 3: HRRN Task Dispatch
  if $readyQueue \neq \varnothing$ **then**
    $candidateTasks \leftarrow$ GetReadyTasks$(readyQueue)$ ▷ Tasks with all dependencies satisfied
    if $candidateTasks \neq \varnothing$ **then**
      $selectedTask \leftarrow$ HRRN_Select$(candidateTasks, currentTime)$
      $assignedVM \leftarrow mappingScheme[selectedTask]$
      if *assignedVM* is available at *currentTime* **then**
        $selectedTask.startTime \leftarrow currentTime$

(Continued)

Algorithm 1 (continued)

$selectedTask.finishTime \leftarrow currentTime$ + estimatedExecutionTime $(selectedTask, assignedVM)$
$selectedTask.assignedVM \leftarrow assignedVM$
$readyQueue \leftarrow readyQueue/selectedTask$
$scheduledTasks \leftarrow scheduledTasks \bigcup selectedTask$
$taskFinishTimes[selectedTask] \leftarrow selectedTask.finishTime$
Update VM availability timeline for *assignedVM*
end if
end if
end if
Step 4: Advance Simulation Time
$nextEventTime \leftarrow$ CalculateNextEventTime$(scheduledTasks, W, taskFinishTimes)$
$currentTime \leftarrow nextEventTime$
end while

Algorithm 2: TSA_GlobalOptimize

Input: ReadyQueue R, VM list V, *maxIterations*, *popSize*
Output: Task-to-VM mapping scheme M

Initialize population P with *popSize* random mappings
$bestFitness \leftarrow \infty$; $bestMapping \leftarrow \varnothing$; $FS \leftarrow \varnothing$; $t \leftarrow 0$
while $t < maxIterations$ **do**
  for each individual $P_i \in P$ **do**
    $fitness_i \leftarrow EvaluateFitness(P_i, R, V)$
    if $fitness_i < bestFitness$ **then**
      $bestFitness \leftarrow fitness_i$; $bestMapping \leftarrow P_i$; $FS \leftarrow P_i$
    end if
  end for
  for each individual P_i in P **do**
    $c_1, c_2, c_3 \leftarrow Random([0,1])$; $r \leftarrow Random([0,1])$
    $M \leftarrow \lfloor P_{min} + c_1 \cdot (P_{max} - P_{min}) \rfloor$
    $F \leftarrow 2 \cdot c_1$; $G \leftarrow c_2 + c_3 - F$; $A \leftarrow G/M$
    $D \leftarrow |FS - r \cdot P_i|$
    if $r \geq 0.5$ **then**
      $P_i^{candidate} \leftarrow FS + A \cdot D$
    else
      $P_i^{candidate} \leftarrow FS - A \cdot D$
    end if
    $P_i \leftarrow (P_i + P_i^{candidate})/(2 + c_1)$
  end for
  $t \leftarrow t + 1$
end while
return *bestMapping*

Algorithm 3: Fitness evaluation for complete schedule

Input: Mapping scheme M, Task set T, VM set V
Output: Total cost of the schedule

$totalRent \leftarrow 0$; $totalPenalty \leftarrow 0$
$vmFinishTime \leftarrow \phi$; $taskFinishTime \leftarrow \phi$; $vmUsageIntervals \leftarrow \phi$
$\varepsilon \leftarrow 1.5$; $\delta \leftarrow 10$ ▷ SLA parameters
for each task $t \in T$ in topological order **do**
    $vm \leftarrow M[t]$
    $earliestStart \leftarrow \max(t.arrivalTime, vmFinishTime[vm])$
    $predecessorFinish \leftarrow \max_{p \in \text{pred}(t)} taskFinishTime[p]$
    $startTime \leftarrow \max(earliestStart, predecessorFinish)$
    $finishTime \leftarrow startTime + t.burstTime/vm.computePower$
    $vmFinishTime[vm] \leftarrow finishTime$; $taskFinishTime[t] \leftarrow finishTime$
    $vmUsageIntervals[vm] \leftarrow vmUsageIntervals[vm] \cup [startTime, finishTime]$
end for
for each VM $v \in V$ used in M **do**
    $totalBusyTime \leftarrow$ calculateTotalBusyTime$(vmUsageIntervals[v])$
    $totalRent \leftarrow totalRent + v.price \times \lceil totalBusyTime/3600 \rceil$
end for
for each task $t \in T$ **do**
    $deadline \leftarrow t.arrivalTime + \varepsilon \times t.burstTime$
    if *taskFinishTime[t] > deadline* **then**
        $totalPenalty \leftarrow totalPenalty + (taskFinishTime[t] - deadline) \times \delta$
    end if
end for
return $totalRent + totalPenalty$

5 Experimental Results

5.1 Experimental Environment Configuration

The experiments are conducted on a Windows 10 machine. Algorithms are implemented in Java (JDK 1.8) and are executed. Custom Java modules for task queue management, the VM modeling, and scheduling control are developed for simulation.

5.2 Experimental Setup and Parameter Settings

5.2.1 Datasets

VM datasets: The VM datasets are sourced from $AmazonEC_2$ [1] and include six configurations (in Table 1) with varying compute, memory, and hourly cost attributes. The number of VM instances is flexible per type. Workflows consist of tasks with unknown patterns and resource requirements until arrival, enabling dynamic allocation to running VMs. This approach enhances resource utilization and adaptability to uncertain, fluctuating workloads.

Table 1: Six types of VM instances configurations.

Type	vCPU	Memory	On-Demand Hourly Rate
m6g.large	2	8 GiB	$0.077
m6g.xlarge	4	16 GiB	$0.154
m6g.2xlarge	8	32 GiB	$0.308
m6g.4xlarge	16	64 GiB	$0.616
m6g.8xlarge	32	128 GiB	$1.232
m6g.12xlarge	48	192 GiB	$1.848

Workflow datasets: Our evaluation uses mixed workflow scenarios to simulate cloud resource competition, integrating four scientific workflows (CyberShake, Inspiral, Montage, Sipht [24]) with distinct structures into three concurrent testing scenarios: Scenario S (30 tasks CyberShake + 30 tasks Montage), Scenario M (50 tasks Inspiral + 50 tasks Sipht), and Scenario L (staggered arrivals of 100 tasks CyberShake, 50 tasks Montage, and 30 tasks Inspiral). This design tests scheduler performance under heterogeneous, dynamic loads, with a consistent $0.24 per hour deadline penalty and system level performance evaluation.

Baseline algorithms: We evaluate the proposed TS-HRRN method against six scheduling algorithms to ensure a comprehensive comparison. This includes the standalone HRRN and TSA schedulers, which serve as baseline components to demonstrate the performance improvement achieved by their integration in our hybrid method. We also compare against four state of the art schedulers: Deadline aware and Cost effective Hybrid Genetic Task Scheduling (DCHG-TS) [21], Ant Colony Optimization (ACO) [22], the Cost Effective Firefly based Algorithm (CEFA) [23], and a Genetic Algorithm with Deep Reinforcement Learning (GA-DRL) [9]. This selection enables a thorough performance assessment across heuristic, metaheuristic, and learning based algorithmic paradigms.

5.2.2 Parameter Settings

To ensure a fair comparison, hyperparameters are configured as follows. Population-based algorithms (standalone TSA, the TS-HRRN, the ACO, the CEFA, the DCHG-TS, and the GA-DRL) used a population size of 100 and 200 iterations. For the GA-DRL, the mutation probability was 0.25, with a DQN agent configured with one hidden layer (16 neurons), replay memory size 200, learning rate 0.01, and discount factor 0.9. The HRRN, as a deterministic heuristic, requires no parameters. For the TSA (original and within the TS-HRRN), velocity bounds are set as $P_{\min} = 1$ and $P_{\max} = 4$, with random factors $c_1, c_2, c_3, r \sim U(0,1)$. Baseline algorithms use original parameters: ACO ($\alpha = 1, \beta = 2, \rho = 0.5$), CEFA ($\gamma = 1.0, \beta = 2.0, \alpha = 0.9$), and DCHG-TS ($P_c = 0.8, P_m = 0.2$). In the fitness function, the SLA penalty factor $\delta = 0.24$ h, and the deadline-relaxation coefficient ε varied over $\{1.00, 1.25, \ldots, 2.25\}$.

5.3 Comparison of Algorithm Costs under Different SLA Relaxation Coefficients

Table 2 reports the cost of the HRRN, the TSA, the ACO, the CEFA, the DCHG-TS, the GA-DRL and our proposed TS-HRRN across combinations of relaxation coefficient and workflow instance scale (Scenario S: 30 CyberShake + 30 Montage, Scenario M: 50 Inspiral + 50 Sipht, Scenario L: 100 CyberShake + 50 Montage + 30 Inspiral). <1.00, S> denotes $\varepsilon = 1.00$ on Scenario S workflow instances. The symbols $(+)$, $(-)$ and $(\approx)$ indicate that, under a Wilcoxon signed rank test at the 0.05 significance level, the result is significantly better, worse, or statistically equivalent to the corresponding baseline algorithm.

Table 2: Average total cost over 30 independent runs.

Scenarios	HRRN	TSA	ACO	CEFA	DCHG-TS	GA-DRL	TS-HRRN
$<1.00, S>$	8.24(2.34)	8.12(2.21)(≈)	35.27(3.23)(−)(−)	185.25(3.34)(−)(−)(−)	18.77(2.12)(−)(−)(+)(+)	4.85(1.20)(+)(+)(+)(+)(+)	5.27(0.43)(+)(+)(+)(+)(+)(≈)
$<1.00, M>$	16.98(5.12)	15.97(4.90)(≈)	78.92(4.33)(−)(−)	455.22(5.11)(−)(−)(−)	28.31(3.82)(−)(−)(+)(+)	8.10(2.50)(+)(+)(+)(+)(+)	8.85(1.89)(+)(+)(+)(+)(+)(≈)
$<1.00, L>$	42.83(8.89)	41.56(8.23)(≈)	225.87(10.01)(−)(−)	1680.09(9.71)(−)(−)(−)	52.33(6.35)(−)(−)(+)(+)	14.25(4.80)(+)(+)(+)(+)(+)	15.53(3.28)(+)(+)(+)(+)(+)(≈)
$<1.25, S>$	7.01(2.26)	7.12(2.33)(≈)	32.08(2.14)(−)(−)	175.54(6.12)(−)(−)(−)	17.74(3.11)(−)(−)(+)(+)	3.95(1.05)(+)(+)(+)(+)(+)	4.31(0.57)(+)(+)(+)(+)(+)(≈)
$<1.25, M>$	15.88(5.98)	14.12(5.21)(≈)	75.36(4.23)(−)(−)	452.12(8.88)(−)(−)(−)	26.22(4.12)(−)(−)(+)(+)	7.15(2.31)(+)(+)(+)(+)(+)	7.85(1.67)(+)(+)(+)(+)(+)(≈)
$<1.25, L>$	40.47(9.18)	38.23(8.90)(≈)	215.84(8.66)(−)(−)	1657.99(11.87)(−)(−)(−)	49.22(6.97)(−)(−)(+)(+)	11.60(3.95)(+)(+)(+)(+)(+)	12.70(2.12)(+)(+)(+)(+)(+)(≈)
$<1.50, S>$	6.98(2.65)	6.46(2.46)(≈)	29.89(2.09)(−)(−)	165.43(5.02)(−)(−)(−)	16.71(3.51)(−)(−)(+)(+)	2.80(0.92)(+)(+)(+)(+)(+)	3.31(0.88)(+)(+)(+)(+)(+)(−)
$<1.50, M>$	14.78(4.27)	13.96(4.95)(≈)	68.80(3.99)(−)(−)	428.73(7.98)(−)(−)(−)	24.14(4.19)(−)(−)(+)(+)	6.10(2.15)(+)(+)(+)(+)(+)	6.85(2.24)(+)(+)(+)(+)(+)(≈)
$<1.50, L>$	38.13(5.98)	37.11(5.27)(≈)	205.82(12.98)(−)(−)	1580.89(12.19)(−)(−)(−)	47.19(8.17)(−)(−)(+)(+)	9.45(4.10)(+)(+)(+)(+)(+)	10.69(3.99)(+)(+)(+)(+)(+)(≈)
$<1.75, S>$	6.45(2.39)	5.89(2.27)(≈)	27.70(2.67)(−)(−)	155.63(2.74)(−)(−)(−)	15.67(2.85)(−)(−)(+)(+)	2.35(0.75)(+)(+)(+)(+)(+)	2.81(0.28)(+)(+)(+)(+)(+)(−)
$<1.75, M>$	13.68(5.95)	12.91(5.15)(≈)	65.24(8.99)(−)(−)	415.05(10.88)(−)(−)(−)	23.07(4.20)(−)(−)(+)(+)	5.25(1.98)(+)(+)(+)(+)(+)	5.85(1.46)(+)(+)(+)(+)(+)(≈)
$<1.75, L>$	36.79(9.06)	35.85(8.89)(≈)	195.79(20.00)(−)(−)	1520.22(20.09)(−)(−)(−)	45.18(9.12)(−)(−)(+)(+)	8.55(3.85)(+)(+)(+)(+)(+)	9.69(2.35)(+)(+)(+)(+)(+)(≈)
$<2.00, S>$	5.91(3.04)	5.87(2.26)(≈)	25.51(1.88)(−)(−)	145.89(2.88)(−)(−)(−)	14.64(2.24)(−)(−)(+)(+)	2.05(0.65)(+)(+)(+)(+)(+)	2.51(0.97)(+)(+)(+)(+)(+)(−)
$<2.00, M>$	12.59(5.99)	11.94(5.95)(≈)	62.69(4.22)(−)(−)	405.45(3.90)(−)(−)(−)	22.01(6.35)(−)(−)(+)(+)	4.60(1.75)(+)(+)(+)(+)(+)	5.15(2.89)(+)(+)(+)(+)(+)(≈)
$<2.00, L>$	35.45(10.95)	34.07(10.32)(≈)	185.77(7.89)(−)(−)	1480.13(5.90)(−)(−)(−)	44.19(15.15)(−)(−)(+)(+)	7.80(3.50)(+)(+)(+)(+)(+)	8.89(4.11)(+)(+)(+)(+)(+)(≈)
$<2.25, S>$	5.48(2.95)	4.88(2.44)(≈)	23.32(1.78)(−)(−)	135.93(2.27)(−)(−)(−)	13.61(3.11)(−)(−)(+)(+)	1.85(0.55)(+)(+)(+)(+)(+)	2.31(0.20)(+)(+)(+)(+)(+)(−)
$<2.25, M>$	11.89(5.93)	11.29(5.67)(≈)	60.13(3.89)(−)(−)	395.39(8.99)(−)(−)(−)	21.18(4.24)(−)(−)(+)(+)	4.25(1.60)(+)(+)(+)(+)(+)	4.75(0.56)(+)(+)(+)(+)(+)(≈)
$<2.25, L>$	34.41(9.01)	33.20(8.99)(≈)	175.78(18.23)(−)(−)	1425.87(10.80)(−)(−)(−)	43.33(8.33)(−)(−)(+)(+)	7.15(3.25)(+)(+)(+)(+)(+)	8.19(1.98)(+)(+)(+)(+)(+)(≈)

Note: ≈: Not significantly different; +: Significantly better; −: Significantly worse.

Results show that the TS-HRRN outperforms its standalone components (the HRRN and the TSA). For example, in scenario $\langle 1.50, M \rangle$, it reduces total cost by 71.6% vs. the DCHG-TS and by 90.0% vs. the ACO. This advantage scales with workflow complexity: in $\langle 1.50, L \rangle$, the TS-HRRN incurs a cost of only 10.69, representing reductions of 94.8% and 77.4% compared to the ACO and DCHG-TS, respectively. Compared to the GA-DRL, the TS-HRRN remains highly competitive; although the GA-DRL achieves a slightly lower cost in some cases (e.g., 9.45 at $\langle 1.50, L \rangle$), the margin is narrow, and the TS-HRRN offers superior computational efficiency and operational simplicity. All improvements over non-learning-based baselines are statistically significant ($p < 0.05$, Wilcoxon rank-sum test). Furthermore, the TS-HRRN's penalty cost decreases with increasing ε, demonstrating its ability to leverage relaxed deadlines. Maintain low costs for all $\varepsilon \geq 1.25$ in large-scale scenarios, showing robustness.

The overall cost comparison across all algorithms and scenarios is synthesized in Fig. 2. A key observation is that TS-HRRN uniquely employs an adaptive strategy: it strictly adheres to deadlines when $\varepsilon < 1.00$, and strategically accepts minimal penalties to leverage low-cost VMs when deadlines are relaxed ($\varepsilon \geq 1.25$), achieving the lowest overall cost. This stands in contrast to the static policies of the HRRN/TSA and the cost-ineffective VM selections of the ACO/CEFA. Compared to the learning-based the GA-DRL, the TS-HRRN attains competitive cost efficiency with significantly lower computational complexity, underscoring its practical advantage.

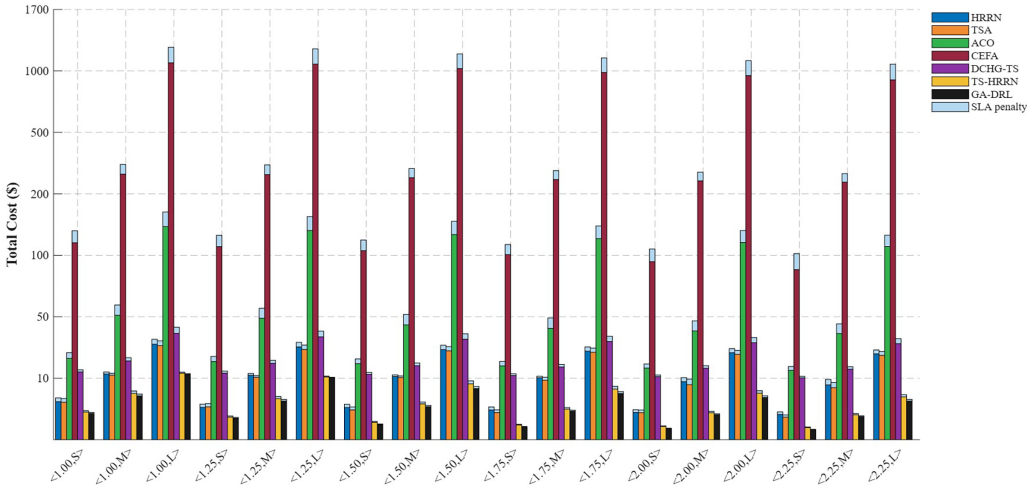


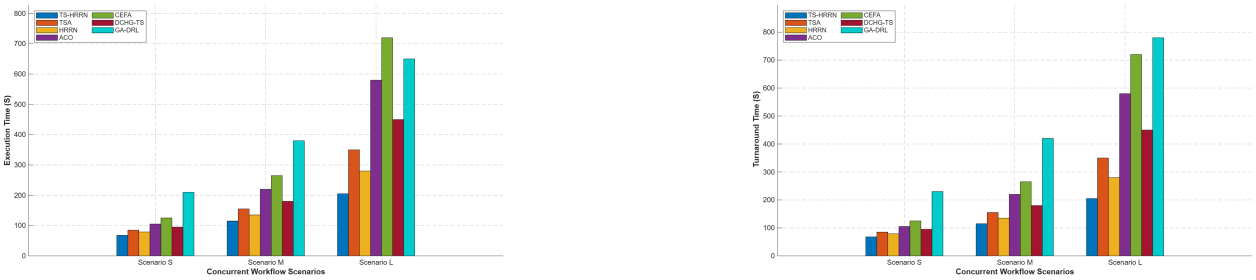
Figure 2: The average of VM fees and SLA penalties of all algorithms.

5.4 Comparative Performance Analysis

Performance Evaluation of TS-HRRN Compared to Other Algorithms

We evaluate the TS-HRRN against its component algorithms and state-of-the-art methods, including the GA-DRL, across three mixed workflow scenarios using the consistent parameter settings detailed in Section 5.2.2.

Fig. 3 shows execution efficiency and turnaround time. The results confirm that integrating the HRRN (for fast dispatch) with the TSA (for global cost optimization) gives the TS-HRRN balanced performance across all metrics. Notably, the TS-HRRN achieves this with approximately three times lower scheduling overhead than the GA-DRL method, demonstrating greater practical efficiency. Consistent gains across scenarios (S, M, L) validate the robustness and adaptability of our approach. Consequently, the method capably handles both transient and macroscopic workload dynamics.



(a) The Execution Time in Mixed Workflow Scenarios

(b) The Turnaround Time in Mixed Workflow Scenarios

Figure 3: Performance comparison in mixed workflow scenarios.

6 Summary

This paper proposes the TS-HRRN, a workflow scheduling method that synergistically combines the TSA with the HRRN policy. Our approach effectively minimizes the rental cost and the SLA penalties for heterogeneous workflows. Extensive evaluation demonstrates that the TS-HRRN reduces the cost by up to 94.8% against meta-heuristic baselines. It also achieves competitive cost-efficiency compared to a learning-based method, while offering superior operational simplicity and stability, confirming its practical value and excellent scalability. In the future, we will deploy the proposed method in fully dynamic cloud settings with auto-scaling and stochastic arrivals. We also plan to extend it to the cloud-edge environment and investigate integration with multi-objective optimization.

Acknowledgement: Not applicable.

Funding Statement: This research was supported by the National Natural Science Foundation of China under Grant 62472264, and the Natural Science Distinguished Youth Foundation of Shandong Province under Grant ZR2025QA13.

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, methodology and writing, Yujie Tian; supervision and project administration, Jing Li; formal analysis, Ming Zhu; review and editing, Cong Liu; data curation, Ziyang Zhang. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are available from the Corresponding Author, Jing Li, upon reasonable request.

Ethics Approval: Not applicable.

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

References

1. Yang Y, Chen G, Ma H, Zhang M. Dual-tree genetic programming for deadline-constrained dynamic workflow scheduling in cloud. In: Service-oriented computing. Cham, Switzerland: Springer Nature; 2022. p. 433–48. doi: 10.1007/978-3-031-20984-0_31.
2. Versluis L, Iosup A. A survey of domains in workflow scheduling in computing infrastructures: community and keyword analysis, emerging trends, and taxonomies. Future Gener Comput Syst. 2021;123(2011):156–77. doi:10.1016/j.future.2021.04.009.
3. Chen W, Deelman E. WorkflowSim: a toolkit for simulating scientific workflows in distributed environments. In: Proceedings of the 2012 IEEE 8th International Conference on E-Science; 2012 Oct 8–12; Chicago, IL, USA. p. 1–8.
4. Fard HM, Prodan R, Fahringer T. Multi-objective list scheduling of workflow applications in distributed computing infrastructures. J Parallel Distribut Comput. 2014;74(3):2152–65. doi:10.1016/j.jpdc.2013.12.004.

5. Mikram H, El Kafhali S, Saadi Y. HEPGA: a new effective hybrid algorithm for scientific workflow scheduling in cloud computing environment. Simul Model Pract Theory. 2024;130(1):102864. doi:10.1016/j.simpat.2023.102864.
6. Arabnejad V, Bubendorfer K, Ng B. Dynamic multi-workflow scheduling: a deadline and cost-aware approach for commercial clouds. Future Gener Comput Syst. 2019;100(1):98–108. doi:10.1016/j.future.2019.04.029.
7. Wang ZJ, Zhan ZH, Yu WJ, Lin Y, Zhang J, Gu TL, et al. Dynamic group learning distributed particle swarm optimization for large-scale optimization and its application in cloud workflow scheduling. IEEE Trans Cybern. 2019;50(6):2715–29. doi:10.1109/tcyb.2019.2933499.
8. Yang Y, Chen G, Ma H, Zhang M, Huang V. Budget and SLA aware dynamic workflow scheduling in cloud computing with heterogeneous resources. In: Proceedings of the 2021 IEEE Congress on Evolutionary Computation (CEC); 2021 Jun 28–Jul 1; Kraków, Poland.
9. Zhang J, Cheng L, Liu C, Zhao Z, Mao Y. Cost-aware scheduling systems for real-time workflows in cloud: an approach based on genetic algorithm and deep reinforcement learning. Expert Syst Appl. 2023;234(1):120972. doi:10.1016/j.eswa.2023.120972.
10. Dong T, Xue F, Xiao C, Zhang J. Workflow scheduling based on deep reinforcement learning in the cloud environment. J Ambient Intell Humaniz Comput. 2021;12(12):1–13. doi:10.1007/s12652-020-02884-1.
11. Faragardi HR, Saleh Sedghpour MR, Fazliahmadi S, Fahringer T, Rasouli N. GRP-HEFT: a budget-constrained resource provisioning scheme for workflow scheduling in IaaS clouds. IEEE Trans Parallel Distrib Syst. 2020;31(6):1239–54. doi:10.1109/tpds.2019.2961098.
12. Ismayilov G, Topcuoglu HR. Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing. Future Gener Comput Syst. 2020;102(5):307–22. doi:10.1016/j.future.2019.08.012.
13. Sanodiya RK, Sharma S, Sharma V. A highest response ratio next (HRRN) algorithm based load balancing policy for cloud computing. Int J Comput Sci Trends Technol. 2013;3(1):72–6.
14. Rimal BP, Maier M. Workflow scheduling in multi-tenant cloud computing environments. IEEE Trans Parallel Distribut Syst. 2016;28(1):290–304. doi:10.1109/tpds.2016.2556668.
15. Ahmed S, Omara FA. An enhanced workflow scheduling algorithm for cloud computing environment. Int J Intell Eng Syst. 2022;15(6):627–46. doi:10.22266/ijies2022.1231.56.
16. Djigal H, Feng J, Lu J, Ge J. IPPTS: an efficient algorithm for scientific workflow scheduling in heterogeneous computing systems. IEEE Trans Parallel Distrib Syst. 2021;32(5):1057–71. doi:10.1109/tpds.2020.3041829.
17. Sun F, Cao J, Lu Z. HEFT-dynamic scheduling algorithm in workflow scheduling. In: Proceedings of the 2022 34th Chinese Control and Decision Conference (CCDC); 2022 Aug 15–17; Hefei, China. p. 4885–90.
18. Chen K, Wang Y, Zhang L, Xie G. A deadline-constrained and cost-minimized approach for workflow scheduling in IaaS clouds. In: Proceedings of the 2022 IEEE 6th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC); 2022 Oct 3–5; Beijing, China. p. 1080–5.
19. Mohammadzadeh A, Masdari M. Scientific workflow scheduling in multi-cloud computing using a hybrid multi-objective optimization algorithm. J Ambient Intell Humaniz Comput. 2023;14(4):3509–29. doi:10.1007/s12652-021-03482-5.
20. Zeedan M, Attiya G, El-Fishawy N. Enhanced hybrid multi-objective workflow scheduling approach based on artificial bee colony in cloud computing. Computing. 2023;105(1):217–47. doi:10.1007/s00607-022-01116-y.
21. Iranmanesh A, Naji HR. DCHG-TS: a deadline-constrained and cost-effective hybrid genetic algorithm for scientific workflow scheduling in cloud computing. Cluster Comput. 2021;24(2):667–81. doi:10.1007/s10586-020-03145-8.
22. Varshney S, Srivastava GMS. Efficient workflow scheduling in fog-cloud environments using ant colony optimization. In: Proceedings of the 2024 Second International Conference on Data Science and Information System (ICDSIS); 2024 May 17–18; Hassan, India. p. 1–5.
23. Chakravarthi K, Shyamala L, Vaidehi V. Cost-effective workflow scheduling approach on cloud under deadline constraint using firefly algorithm. Appl Intell. 2021;51(3):1629–44. doi:10.1007/s10489-020-01875-1.
24. Juve G, Chervenak A, Deelman E, Bharathi S, Mehta G, Vahi K. Characterizing and profiling scientific workflows. Future Gener Comput Syst. 2013;29(3):682–92. doi:10.1016/j.future.2012.08.015.