IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 5, October 2026
Impact Factor: 8.2

Reinforcement Learning Approach for Optimizing Cloud Resource Utilization with Load Balancing

Miss. Sonawane Rutuja Sitaram[1], Miss. P. V. Kapse[2], Prof. J. R. Mahajan[3]
[1] Student, [2] Guide,[3] Head of Department (HOD)
Department of Artificial Intelligence and Data Science
Adsul Technical Campus, Chas, Ahilyanagar, India

Abstract: *Cloud computing provides flexible access to processing, storage, and networking resources, but efficient use of these resources remains a difficult management problem. Workloads vary in size, arrival rate, execution time, and resource demand, while virtual machines and containers may differ in capacity and current utilization. A static scheduler can perform well under one workload and poorly when the workload changes. This paper presents a reinforcement learning (RL)-based framework for cloud task scheduling and load balancing. The proposed controller observes system conditions, selects a suitable placement or scheduling action, and receives feedback based on execution delay, resource utilization, queue length, migration cost, and service-level constraints. The design uses a Markov decision process to describe the scheduling problem and considers Q-learning or a deep Q-network as possible policy-learning methods. A practical evaluation plan is defined using repeatable workload traces and baseline schedulers such as First Come First Served, Round Robin, Min-Min, Max-Min, and Minimum Completion Time. The paper specifies metrics, experimental controls, and implementation considerations so that the framework can be evaluated without claiming results that have not yet been measured. The main contribution is a structured, reproducible design for learning-based load balancing that accounts for both performance and operational overhead.*

Keywords: Cloud computing, reinforcement learning, task scheduling, load balancing, resource utilization, Q-learning, virtual machines, workload management.

I. INTRODUCTION

Cloud computing allows organizations to provision compute and storage capacity on demand rather than maintaining every resource locally. This model supports applications ranging from web services and data analytics to scientific workflows and enterprise systems. However, the flexibility of cloud infrastructure introduces a continuous control problem: incoming tasks must be assigned to available resources while demand and resource conditions change over time.

A scheduler must consider more than whether a virtual machine is available. It should account for processor utilization, memory pressure, queue length, estimated task duration, data-transfer requirements, energy use, and service-level objectives. If tasks accumulate on a small number of servers, response times increase and some resources become overloaded while others remain underused. Conversely, aggressive distribution can add communication overhead, increase migration traffic, or activate unnecessary machines. Effective load balancing therefore seeks a practical compromise between performance, fairness, cost, and stability.

Traditional scheduling policies are attractive because they are simple and easy to implement. First Come First Served (FCFS) follows arrival order, Round Robin (RR) distributes assignments cyclically, and Min-Min or Max-Min use estimated completion times to choose tasks. These approaches are useful baselines, but their decisions generally depend on a fixed rule or a limited snapshot of the system. When workloads are heterogeneous or bursty, a fixed policy may not adapt adequately to new conditions.

Copyright to IJARSCT
www.ijarsct.co.in
DOI: 10.48175/IJARSCT-40157

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 5, October 2026
Impact Factor: 8.2

Reinforcement learning offers an alternative. An agent learns a policy by interacting with an environment, observing the consequences of its actions, and improving decisions from reward feedback. For cloud scheduling, the environment can represent a cluster of virtual machines or containers, while actions correspond to task placement, resource selection, or carefully constrained migration. The reward can encourage low completion time and balanced utilization while penalizing overload, deadline violations, energy consumption, and excessive movement of running workloads.

This paper proposes an RL-based load-balancing framework and a method for evaluating it. It is a design and evaluation proposal rather than a report of completed experiments. No numerical performance improvement is asserted without implementation and measurement. The objective is to provide a clear system model, learning formulation, algorithmic workflow, and reproducible evaluation plan that can be implemented in a simulator or a test cluster.

II. LITERATURE REVIEW

Several researchers have explored task scheduling and load balancing to improve resource utilization in cloud computing.

Lahande et al. [1] studied cloud load balancing and resource utilization using different scheduling techniques. Their work highlights the importance of distributing tasks efficiently across available resources.

Kruekaew and Kimpan [2] proposed a hybrid approach combining the Artificial Bee Colony algorithm with reinforcement learning for multi-objective task scheduling. Their study focuses on improving scheduling efficiency and balancing workloads.

Vengerov [3] investigated reinforcement learning for dynamic resource allocation. This work provides a foundation for using learning-based techniques to make resource management decisions in changing environments.

Chen and Deelman [4] introduced WorkflowSim, a simulation toolkit for evaluating scientific workflows in distributed computing environments. Such tools can support controlled testing of scheduling algorithms.

Research Gap: Traditional scheduling algorithms may struggle with changing workloads and heterogeneous resources. Reinforcement learning offers a potential adaptive solution, but its effectiveness must be verified through experiments comparing execution time, resource utilization, and load balancing.

III. METHODOLOGY

A. Research Design

This study proposes a reinforcement learning approach to improve resource utilization and load balancing in cloud computing. The main objective is to examine how a learning-based scheduler can distribute tasks across available resources while considering execution time, workload distribution, and resource availability.

The proposed approach will be evaluated by comparing it with traditional scheduling algorithms, including First Come First Served (FCFS), Round Robin (RR), Min-Min, Max-Min, and Minimum Completion Time (MCT). The same workload and resource configuration will be used for each method to ensure a fair comparison.

The evaluation will begin in a simulated cloud environment. If the results are promising, the approach can subsequently be tested on a real cloud or container-based platform.

B. Proposed System

The proposed system consists of a resource monitoring component, a task scheduler, a reinforcement learning agent, and a feedback mechanism.

The monitoring component collects information about CPU utilization, memory consumption, waiting tasks, and resource availability. The scheduler uses this information to determine where an incoming task should be executed.

The reinforcement learning agent observes the current system condition and selects a suitable resource. After the task has been processed, the system evaluates the outcome and provides feedback to the agent. Over repeated training episodes, the agent can learn which scheduling decisions are more effective under different workload conditions.

This approach is intended to make scheduling more adaptive than relying entirely on a fixed scheduling rule.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 5, October 2026
Impact Factor: 8.2

C. Reinforcement Learning Model

The scheduling problem is represented using three main elements: state, action, and reward.

State: The state describes the current cloud environment. It includes resource utilization, queue length, available processing capacity, and information about the task waiting to be scheduled.

Action: The action represents the selection of an appropriate virtual machine or computing resource for a task.

Resources that do not meet the task's requirements will be excluded from the available choices.

Reward: The reward provides feedback about the quality of a scheduling decision. The proposed reward function will consider task completion time, load imbalance, resource overload, and scheduling overhead. Suitable reward weights will be selected during training and validation.

The objective is to develop a policy that makes effective scheduling decisions while balancing resource utilization and application performance.

D. Experimental Procedure

The experiment will follow a structured process:

Configure the cloud resources and prepare the workload. Collect resource information and task characteristics.

Provide the observed system state to the reinforcement learning agent. Select a suitable resource for the next eligible task.

Execute the task and record its completion time and resource usage. Calculate the reward based on the observed outcome.

Update the learning model during training.

Evaluate the trained model using workloads that were not used during training. Compare its performance with the traditional scheduling algorithms.

The final evaluation will use a fixed, trained policy so that test results do not influence further model training.

E. Performance Evaluation Metrics

The proposed method will be evaluated using several performance measures. Average completion time: Measures the average time required for tasks to finish. Makespan: Measures the total time needed to complete the entire workload.

Resource utilization: Shows how effectively the available computing resources are being used. Load imbalance: Measures differences in workload across resources.

Scheduling overhead: Measures the time and computing effort required to make scheduling decisions.

Deadline-miss rate: Measures how frequently deadline-constrained tasks finish late, if deadlines are included in the experiment.

Considering multiple metrics will help determine whether the proposed method offers a practical improvement rather than improving one measure at the expense of another.

IV. BACKGROUND AND RELATED WORK

A. Cloud Resource Management

Cloud resource management covers provisioning, placement, scaling, scheduling, monitoring, and recovery. Virtualization enables multiple workloads to share physical infrastructure, but sharing also creates interference. A CPU-intensive task can compete with another task for processor time, while memory-intensive workloads can trigger swapping or eviction. Network-intensive jobs may be limited by data movement rather than compute capacity. A scheduler that uses only CPU load can therefore misjudge the true cost of a placement.

Resource managers typically separate monitoring from decision-making. Monitoring collects utilization and queue statistics, while the scheduler converts these observations into placements or scaling actions. The quality of the scheduler depends on the freshness and accuracy of the measurements. Excessively frequent monitoring adds overhead;

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 5, October 2026
Impact Factor: 8.2

infrequent monitoring may hide short-lived spikes. An RL controller should therefore be evaluated not only on its policy but also on the information available to it.

B. Traditional Scheduling Policies

FCFS is straightforward and preserves arrival order, but a long task at the front of a queue can delay shorter tasks. RR rotates among resources or queues and can improve distribution when capacities are similar, although equal rotation is not necessarily fair when machines have different speeds. Min-Min chooses a task with a small estimated completion time, which may favor short jobs. Max-Min tends to schedule a task with a large completion time first and can protect long jobs from indefinite postponement, but may delay smaller tasks. Minimum Completion Time (MCT) assigns each task to the resource predicted to finish it soonest. The quality of these methods depends on how accurately task and resource characteristics are estimated.

Heuristics remain valuable because they are transparent, computationally inexpensive, and often require little training data. They should not be treated as weak straw-man baselines. A learning-based scheduler must demonstrate measurable benefits under consistent experimental conditions and should report cases where simpler methods perform as well or better.

C. Reinforcement Learning for Scheduling

In reinforcement learning, an agent observes a state, selects an action, receives a reward, and transitions to a new state. The goal is to learn a policy that maximizes expected cumulative reward. Q-learning estimates the value of taking an action in a state and following the best policy thereafter. It is practical for small or discretized state-action spaces. When observations contain many continuous variables, a deep Q-network (DQN) can approximate action values using a neural network, although it requires careful training and validation.

Prior research has explored RL for resource allocation and task scheduling. Vengerov examined reinforcement learning for dynamic resource allocation [3]. Kruekaew and Kimpan investigated a hybrid artificial bee colony and reinforcement learning approach to multi-objective task scheduling [2]. Lahande et al. studied load balancing in cloud computing and discussed reinforcement learning as a promising direction for adaptive scheduling [1]. These works motivate the proposed design, but their reported setups and metrics should not be assumed to apply directly to every cloud platform.

D. Workflow Simulation

Scientific workflow simulators can model task dependencies, resource execution, and scheduling outcomes without requiring a large physical cluster. WorkflowSim, for example, was introduced as a toolkit for simulating scientific workflows in distributed environments [4]. A simulator can make comparisons repeatable and reduce cost, but its results depend on the accuracy of task-duration estimates, network assumptions, failure models, and resource configurations. Simulation is therefore a useful first step, not a substitute for later validation on a real platform.

E. Research Gap and Design Motivation

The practical challenge is not simply to apply an RL algorithm; it is to define a state, action space, and reward that reflect real operational goals. A reward that focuses only on average execution time may overload a subset of machines. A reward that maximizes utilization may leave too little headroom for bursts. A policy trained on a narrow workload may fail when task sizes, arrival rates, or machine capacities change. The proposed framework addresses these concerns by explicitly including multiple metrics, comparing against strong baselines, and separating training workloads from evaluation workloads.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 5, October 2026
Impact Factor: 8.2

V. PROPOSED SYSTEM ARCHITECTURE

The proposed architecture consists of a workload source, a monitoring layer, a state builder, an RL scheduler, an execution layer, and a feedback collector. These components are separated so that the learning method can be changed without rewriting the resource monitor or task executor. The scheduler may run centrally for a small cluster or be integrated with an existing orchestration service for a larger deployment.

A. Workload Source

The workload source produces tasks with attributes such as arrival time, estimated CPU demand, memory requirement, expected duration, input-data size, priority, and deadline where applicable. For a workflow, task dependencies must also be represented so that a task is not dispatched before its prerequisites finish. Workloads may be drawn from publicly documented traces, generated using controlled distributions, or replayed from anonymized logs. The source and all preprocessing decisions should be documented to make the experiment reproducible.

B. Monitoring Layer

The monitor collects per-resource CPU utilization, memory utilization, queue length, active-task count, available capacity, estimated completion time, and relevant network measurements. If energy use is included, it should be measured or estimated consistently across all compared policies. The monitoring interval should be fixed during each experiment, and missing or delayed readings should be handled explicitly rather than silently replaced with ideal values.

C. State Builder

The state builder converts raw monitoring data into a structured observation. A compact state can include normalized utilization values, queue lengths, the characteristics of the task being scheduled, and summary statistics such as average load and load variance. Normalization is important because CPU percentages, memory sizes, and execution times use different scales. For variable cluster sizes, the state representation should either use a fixed maximum number of resources with masks or use aggregate features that preserve relevant differences between resources.

D. RL Scheduler

The scheduler receives the state and selects a feasible action. In a simple version, an action is the choice of one resource for the next task. In an extended version, the action may also choose whether to defer a task briefly or request a scaling operation, but these additional actions enlarge the learning problem and should be introduced only when necessary. A feasibility filter rejects actions that violate memory limits, dependency constraints, or resource availability. The selected

E. Execution and Feedback

The execution layer submits the task to the chosen virtual machine or container and tracks its lifecycle. When the task completes, fails, or times out, the feedback collector records the observed outcome. The reward can be calculated immediately for each dispatch decision or after a task finishes. Delayed rewards are realistic but make credit assignment more difficult. The system should log the state, action, reward components, completion status, and policy version so that results can be audited.

F. Operational Safety and Fallback

An RL policy should not be allowed to bypass platform safety rules. The scheduler must enforce capacity limits, authentication boundaries, task dependencies, and deadline policies independently of the learned model. A fallback policy such as MCT or RR should be available if the model is unavailable, confidence is low, or observations become stale. In production, the policy should first run in shadow mode, where its decisions are recorded but not executed, before being permitted to affect live workloads.

VI. REINFORCEMENT LEARNING FORMULATION

The scheduling process can be represented as a Markov decision process (MDP), defined by a state space S, action space A, transition dynamics, reward function R, and discount factor γ. At decision step t, the agent observes state s_t,

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 5, October 2026 Impact Factor: 8.2

chooses action a_t, receives reward r_t, and observes the next state s_(t+1). The learning objective is to find a policy π that maximizes expected discounted return:

$G_t = \Sigma (k=0 \text{ to } \infty)\ \gamma^k r_{(t+k)}$

Here, γ controls the relative importance of future outcomes. A value near zero emphasizes immediate reward, whereas a value closer to one gives more weight to longer-term effects such as queue buildup and repeated overload.

A. State Space

A possible observation at time t is:

$s_t = [u_1 \ldots u_n, m_1 \ldots m_n, q_1 \ldots q_n, c_1 \ldots c_n, x_t]$

where u_i is CPU utilization for resource i, m_i is memory utilization, q_i is queue length, c_i is an estimate of available capacity or completion time, and x_t describes the next task. Task features may include estimated CPU demand, memory demand, data size, priority, and deadline slack. All values should be normalized or encoded consistently. State design should avoid including information that would not be available at decision time, because that would create data leakage in evaluation.

B. Action Space

For a cluster of n eligible resources, the simplest action space is A = {1, 2, …, n}, where action i assigns the next task to resource i. If some resources cannot run a task, those actions are masked before selection. This keeps the policy focused on valid decisions and reduces the chance that training is dominated by impossible placements. A future version may add scale-out or scale-in actions, but scaling should be modeled with realistic startup delays and cost.

C. Reward Function

Reward design determines what the agent learns to optimize. A proposed reward is a weighted combination of normalized performance measures:

$r_t = -\alpha T_t - \beta I_t - \delta O_t - \eta E_t + \lambda B_t$

T_t represents normalized task delay or completion time, I_t represents imbalance across eligible resources, O_t represents overload or constraint violations, E_t represents energy or scheduling overhead, and B_t represents a benefit such as meeting a deadline or reducing queue backlog. The weights α, β, δ, η, and λ are non-negative design parameters. Their values must be selected through training and validation, not adjusted after inspecting the final test results.

A common imbalance measure is the variance of resource utilization. If u_i is utilization for resource i and ū is the mean utilization, then imbalance may be defined as:

$I_t = (1/n)\ \Sigma(i=1 \text{ to } n)\ (u_i - \bar{u})^2$

Variance is easy to calculate but should be interpreted carefully: perfectly equal utilization is not always optimal when resources have different capacities or when some tasks have specialized requirements. Capacity-normalized load or predicted completion time may be more appropriate in heterogeneous clusters. The reward should also include penalties for rejected tasks, failed dispatches, and excessive migration if these are part of the system.

D. Q-Learning Baseline

For a discretized state space, tabular Q-learning updates the value estimate after each transition:

$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha_{lr} [r_t + \gamma \max_a Q(s_{(t+1)},a) - Q(s_t,a_t)]$

The learning rate α_lr controls the size of each update. The agent can use an ε-greedy action policy: with probability ε it explores a feasible action, and otherwise it chooses the action with the highest estimated value. Exploration should be reduced gradually during training, while the final evaluation should use a fixed policy and should not continue learning from test outcomes.

E. Deep Q-Network Option

When the state is too large for a table, a DQN can approximate Q(s,a) with a neural network. Experience replay stores transitions and trains the network on shuffled mini-batches, while a target network can improve training stability. DQN is not automatically better than tabular Q-learning: it adds hyperparameters, computation, and risks of unstable

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 5, October 2026 Impact Factor: 8.2

learning. The simplest method that fits the state space should be used first, with more complex models introduced only when justified by measured limitations.

F. Training and Generalization

Training episodes should cover a range of task sizes, arrival patterns, and resource configurations. A training workload must be separated from validation and test workloads. Validation data can be used for reward-weight selection and hyperparameter tuning; the test set should be used only for final comparison. Multiple random seeds are necessary because exploration and workload generation can produce variable results. A policy that performs well on one synthetic pattern may not generalize to bursty, heterogeneous, or dependency-constrained workloads.

VII. SCHEDULING WORKFLOW AND ALGORITHM

The scheduling loop is designed to be simple enough to reproduce and modular enough to extend.

1**. Initialize**: Load the resource configuration, task source, monitoring interval, reward weights, and model parameters. Verify that all resources are eligible and that the fallback scheduler is available.

2. Observe: Collect the latest resource metrics and the attributes of the task awaiting placement. Reject stale or invalid observations.

3. Construct state: Normalize the measurements and encode the task and cluster state using the fixed state representation.

4. Filter actions: Remove resources that lack required capacity or violate task dependencies and policy constraints.

5. Select action: During training, choose a feasible action using the exploration policy; during evaluation, use the frozen learned policy.

6. Dispatch: Send the task to the selected resource and record the decision, timestamp, and model version.

7. Observe outcome: Collect completion time, queue changes, failures, resource usage, and any overhead attributable to the decision.

8. Calculate reward: Compute the reward using the predefined formula and retain the transition for learning if the run is a training run.

9. Update policy: Apply the learning update only during training. Do not update the policy during final baseline comparison.

10. Repeat: Continue until the workload is completed or the configured experiment horizon is reached.

A. Pseudocode

```
For each episode:
  Reset workload and resource state
  For each scheduling decision:
    Observe current state s
    Build feasible action set A_valid
    Choose a from A_valid using policy π
    Dispatch the next eligible task using a
    Observe next state s' and outcome
    Compute reward r from predefined metrics
    If training: update Q-function using (s, a, r, s')
  End decision loop
End episode
Evaluate frozen policy on unseen workloads
```

B. Complexity Considerations

If the controller evaluates n resources for each task, a straightforward action filter and selection step is generally O(n) per decision, excluding the cost of neural-network inference. A DQN inference cost depends on the network architecture but is usually bounded and can be measured directly. The main practical overhead may come from collecting metrics, estimating task completion times, and moving workloads rather than selecting an action. The evaluation should report scheduling latency so that a more accurate policy is not accepted if its decision time becomes excessive.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 5, October 2026
Impact Factor: 8.2

VIII. EXPERIMENTAL DESIGN AND EVALUATION PLAN

A fair evaluation must make it possible to distinguish the effect of the scheduling policy from the effect of the workload, machine configuration, or random seed. The following plan describes how the framework should be tested. It is not a claim that these experiments have already been executed.

A. Experimental Environment

The first stage can use a simulator such as WorkflowSim or a custom discrete-event model with documented assumptions. The environment should define the number and capacities of resources, task-arrival process, task-duration distribution, memory demands, network costs, and any workflow dependencies. At least one homogeneous configuration and one heterogeneous configuration should be included. A later stage can replay the same workload on a small cloud testbed or container cluster to check whether simulation conclusions hold in practice.

B. Workload Scenarios

- Steady load: tasks arrive at a relatively stable rate, providing a baseline for normal operating conditions.
- Bursty load: arrival rates rise sharply for short intervals, testing response to sudden demand.
- Heterogeneous tasks: short and long tasks are mixed to expose queueing and starvation effects.
- Heterogeneous resources: machines differ in CPU and memory capacity, testing capacity-aware placement.
- Deadline-sensitive tasks: a subset of tasks has deadlines or priority levels, testing service-level performance.
- Resource disturbance: one or more resources become slower or temporarily unavailable, testing robustness and fallback behavior.

C. Baseline Methods

The proposed policy should be compared with FCFS, RR, Min-Min, Max-Min, and MCT where the environment supports their assumptions. Each baseline should receive the same workload, resource inventory, task estimates, and failure conditions. If a baseline uses estimated execution time, the RL method should not receive privileged information unavailable to the baseline unless the study explicitly evaluates that difference. A random feasible scheduler may also be included as a diagnostic baseline, but it should not replace stronger heuristics.

D. Metrics

The primary metrics should include average task completion time, makespan, resource utilization, load imbalance, queueing delay, deadline-miss rate, and scheduling overhead. Depending on the use case, energy consumption, task failure rate, and cost per completed task may also be relevant. Every metric needs a clear definition before the experiments begin.

- Average completion time: the mean elapsed time from task submission to completion.
- Makespan: the elapsed time between the start of the workload and the completion of its last task.
- Mean utilization: average resource utilization over the observation interval, reported alongside per-resource values.
- Load imbalance: a predefined dispersion measure across capacity-normalized resource loads.
- Deadline-miss rate: the fraction of deadline-constrained tasks completed after their deadlines.
- Scheduling overhead: time and compute spent collecting state, selecting actions, and updating the policy.
- Failure rate: the fraction of tasks that fail, are rejected, or exceed the configured timeout.

E. Experimental Controls

Each method should be tested using identical workload traces and repeated random seeds. The report should provide the number of runs, confidence intervals or standard deviations, and the method used for statistical comparison. Training time should be reported separately from inference time. The RL policy should be frozen during test runs, and no test outcomes should be used to tune reward weights or hyperparameters. Ablation studies should remove one reward component at a time to estimate whether each component contributes useful behavior.

F. Reporting Results Transparently

Results should be shown in tables and plots with units, workload descriptions, and uncertainty indicators. A percentage improvement should be calculated against a clearly identified baseline and metric. If the RL method improves average

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 5, October 2026
Impact Factor: 8.2

completion time but worsens deadline-miss rate or overhead, both outcomes must be reported. Negative or inconclusive findings are useful because they indicate the workload conditions under which learning does not provide a clear advantage.

IX. RESULTS TEMPLATE AND INTERPRETATION FRAMEWORK

Since the framework described in this paper has not been experimentally validated within this draft, numerical results are intentionally not invented. After implementation, the following table can be populated with measured values from repeated runs. Use the same units and statistical summary for all methods.

Table. I

Method	Avg. completion time	Makespan	Mean utilization	Imbalance	Scheduling overhead
FCFS	To be measured	To be measured	To be measured	To be measured	To be measured
Round Robin	To be measured	To be measured	To be measured	To be measured	To be measured
Min-Min	To be measured	To be measured	To be measured	To be measured	To be measured
Max-Min	To be measured	To be measured	To be measured	To be measured	To be measured
MCT	To be measured	To be measured	To be measured	To be measured	To be measured
Proposed RL	To be measured	To be measured	To be measured	To be measured	To be measured

A. How to Interpret the Outcomes

A lower makespan indicates that the full workload completed sooner, but it does not necessarily mean that every task experienced low delay. Average completion time should therefore be read alongside tail latency or deadline-miss rate when these are important. High mean utilization can indicate effective resource use, but utilization near saturation can also increase queueing and reduce resilience to bursts. The goal is not to maximize every metric independently; it is to identify a balanced operating point that satisfies the application's requirements.

If RL reduces imbalance but does not improve completion time, the reward may be overemphasizing uniformity or the cluster may already be well balanced under a simple baseline. If training performance is strong but test performance is weak, the policy may be overfitting the training workload. If improvements disappear when scheduling overhead is included, the policy may be too computationally expensive for the decision interval. These outcomes should lead to model or reward redesign rather than selective reporting.

B. Ablation and Sensitivity Analysis

An ablation study should compare the complete reward against variants that omit imbalance, deadline, energy, or overhead terms. Sensitivity analysis should vary reward weights, exploration schedule, learning rate, discount factor, and monitoring interval within documented ranges. The purpose is to identify whether performance depends on a narrow parameter setting. A robust policy should remain reasonably effective across a useful range of settings and should degrade gracefully when conditions differ from training.

X. DISCUSSION

A. Expected Benefits

An RL-based controller may learn interactions that are difficult to encode in a fixed rule, particularly when task characteristics and resource conditions vary over time. It can use feedback from previous scheduling decisions to adjust

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 5, October 2026
Impact Factor: 8.2

placement behavior, and it can optimize a combined objective rather than following a single static heuristic. When trained

on varied workloads, the policy may learn to avoid sending tasks to resources that repeatedly develop long queues or high contention.

The framework is also modular. The state builder, reward function, action filter, and learning algorithm can be evaluated independently. This separation helps researchers identify whether poor performance comes from inaccurate monitoring, an inadequate state representation, a reward that encodes the wrong priorities, or limitations in the learning algorithm.

B. Limitations

RL requires representative experience, and collecting that experience can be expensive. In a real system, exploratory actions may affect users, so training directly on production traffic may be inappropriate. Simulation reduces risk but introduces a sim-to-real gap. If task duration or network delay is modeled inaccurately, the learned policy may prefer placements that are not efficient in deployment.

Reward design is another major limitation. A weighted sum is easy to implement, but selecting weights can be subjective and may hide trade-offs. A policy can exploit gaps in the reward definition—for example, improving utilization while increasing tail latency. Constraint-based designs or explicit service-level limits may be safer than relying only on penalty weights.

Scalability and interpretability also require attention. A large action space can make learning slower, while neural-network policies may be difficult for operators to explain. Model updates can introduce regressions, and monitoring data may be incomplete or delayed. These limitations support the need for baseline fallbacks, policy versioning, offline evaluation, and staged deployment.

C. Deployment Considerations

A practical deployment should begin with offline training and evaluation on recorded or simulated workloads. The model can then run in shadow mode, allowing its choices to be compared with actual scheduler decisions without affecting task placement. A limited canary deployment can follow if shadow results are stable and the policy respects all constraints. Operators should be able to disable the policy quickly, roll back to a known baseline, and inspect the reason for a scheduling decision.

The controller should also consider fairness across tenants. An objective that minimizes cluster-wide completion time may repeatedly disadvantage low-priority users or long-running jobs. Tenant quotas, priority rules, and service-level objectives should be enforced outside the learned policy or encoded as explicit constraints. Security-sensitive workloads should not be moved across trust boundaries simply because a model predicts lower execution time.

D. Future Work

Future research can extend the framework to multi-agent scheduling, autoscaling, energy-aware placement, and workload prediction. A multi-objective approach could expose a Pareto frontier instead of collapsing every goal into one reward. Constrained RL may be useful when deadline or capacity requirements must be respected. Further work should also investigate transfer learning between clusters, safe online adaptation, explainability, and performance under failures or adversarially unusual workload patterns.

XI. CONCLUSION

This paper has presented a design for reinforcement-learning-based cloud task scheduling and load balancing. The proposed architecture connects monitoring, state construction, feasible-action filtering, policy selection, execution, and feedback. The MDP formulation describes how resource utilization, queueing, completion time, and operational costs

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 5, October 2026
Impact Factor: 8.2

can guide a learned policy. A reproducible evaluation plan has been outlined using established scheduling baselines, diverse workload scenarios, repeated runs, and transparent metrics.

The framework should be viewed as a testable proposal, not as evidence that reinforcement learning necessarily outperforms traditional schedulers. Its value must be established through implementation, fair comparison, and validation on workloads that were not used for training. By including safety constraints, a fallback scheduler, and explicit reporting requirements, the design aims to support a careful path from simulation to practical cloud deployment.

XII. ACKNOWLEDGMENT

The author may acknowledge institutional guidance, laboratory resources, funding, or technical assistance here.
Remove this section or revise it to reflect actual support; do not include acknowledgments that are not applicable.

REFERENCES

[1] P. V. Lahande et al., "Reinforcement Learning Approach for Optimizing Cloud Resource Utilization With Load Balancing," IEEE Access, vol. 11, pp. 127567–127577, 2023, doi: 10.1109/ACCESS.2023.3329557.

[2] B. Kruekaew and W. Kimpan, "Multi-objective task scheduling optimization for load balancing in cloud computing environment using hybrid artificial bee colony algorithm with reinforcement learning," IEEE Access, vol. 10, pp. 17803–17818, 2022.

[3] D. Vengerov, "A reinforcement learning approach to dynamic resource allocation," Engineering Applications of Artificial Intelligence, vol. 20, no. 3, pp. 383–390, 2007.

[4] W. Chen and E. Deelman, "WorkflowSim: A toolkit for simulating scientific workflows in distributed environments," in Proc. IEEE 8th International Conference on E-Science, 2012, pp. 1–B. COMPLEXITY