A Hybrid Load Balancing Method Using Q-Learning for Quality of Service Optimization in Cloud Environments

Saketh Pabbu (S20230010169), Nerella Venkata Sriram (S20230010164), Someswarkumar Balam (S20230010230)
Indian Institute of Information Technology Sri City
Department of Computer Science and Engineering
Chittoor, Andhra Pradesh

***Abstract*—This paper presents a novel hybrid load balancing system that addresses the overhead dilemma in cloud computing environments. Traditional load balancing algorithms face a trade-off between computational overhead and Quality of Service (QoS) optimization. Static methods like Round Robin offer low overhead but poor load distribution, while dynamic methods like Ant Colony Optimization provide excellent performance at high computational cost.**

Our proposed solution employs Q-learning to intelligently switch between Round Robin and Ant Colony Optimization based on real-time system state. The system uses a 3×3×2 Q-table to represent load states (low/medium/high), balance states (poor/good/excellent), and algorithm choices (RR/ACO). Through -greedy exploration and Bellman equation-based learning, the hybrid agent achieves optimal economic/performance trade-off.

Experimental results demonstrate that the hybrid approach achieves superior QoS compared to individual algorithms while maintaining computational efficiency. The system was implemented using Python, SimPy for discrete-event simulation, and deployed on AWS infrastructure with real-time monitoring via CloudWatch and persistent storage in DynamoDB.

***Index Terms*—Load Balancing, Q-Learning, Ant Colony Optimization, Cloud Computing, Quality of Service, Hybrid Algorithms**

I. Introduction

Cloud computing environments require efficient load balancing to ensure optimal resource utilization and Quality of Service (QoS) for users. Load balancing distributes incoming tasks across multiple servers to prevent any single server from becoming a bottleneck while maximizing overall system performance.

Traditional load balancing algorithms can be categorized into static and dynamic approaches. Static methods, such as Round Robin (RR), distribute tasks in a cyclic manner without considering server states. While computationally inexpensive, these methods often result in poor load distribution, especially under heterogeneous workloads. Dynamic methods, such as Ant Colony Optimization (ACO), adapt to real-time system conditions but incur significant computational overhead.

This paper addresses the "overhead dilemma" by proposing a hybrid load balancing system that combines the strengths of both approaches. Our solution employs Q-learning to intelligently select between RR and ACO based on system load conditions. The hybrid agent learns optimal switching strategies through reinforcement learning, achieving superior QoS while minimizing computational overhead.

A. Problem Statement

The core challenge in cloud load balancing is the trade-off between:

- **Performance vs. Overhead**: Dynamic algorithms provide better QoS but consume more computational resources
- **Adaptability vs. Efficiency**: Static methods are efficient but cannot adapt to changing workloads
- **QoS vs. Cost**: High-performance algorithms increase operational costs

B. Contributions

Our main contributions include:

1) A Q-learning based hybrid load balancing framework that adaptively switches between RR and ACO
2) Comprehensive evaluation on AWS infrastructure with real-time monitoring
3) Demonstration of superior economic/performance trade-off compared to individual algorithms
4) Open-source implementation using Python and SimPy for reproducible research

II. Related Work

Several researchers have explored hybrid load balancing approaches to address the overhead dilemma. Shrimal et al. [1] proposed a hybrid method combining static and dynamic algorithms for QoS optimization in cloud environments. Their work demonstrated the potential of adaptive algorithms but lacked reinforcement learning components.

Ant Colony Optimization has been widely applied to load balancing problems. Dorigo et al. [**?**] introduced ACO as a meta-heuristic inspired by ant foraging behavior. Subsequent works [**?**], [**?**] applied ACO to cloud load balancing, showing improved performance over traditional methods but with increased computational complexity.

Q-learning and reinforcement learning have been explored for adaptive load balancing. Tesauro et al. [**?**] demonstrated reinforcement learning for autonomic computing. More recent

works [**?**], [**?**] applied deep reinforcement learning to load balancing, achieving adaptive behavior but requiring significant computational resources.

Hybrid approaches combining multiple algorithms have shown promise. Kaur et al. [**?**] proposed a hybrid of Round Robin and Least Connection algorithms. However, these methods lack the intelligent adaptation provided by reinforcement learning.

Our work extends these approaches by integrating Q-learning with ACO and RR, providing adaptive algorithm selection based on real-time system state while maintaining computational efficiency.

III. Proposed Approach

Our hybrid load balancing system consists of three main components: Round Robin, Ant Colony Optimization, and a Q-learning agent that selects between them. The system operates in a cloud environment with multiple servers, each characterized by utilization, response time, and current load.

A. System Architecture

The architecture comprises five main components:

- **Load Balancer**: Receives incoming tasks and assigns them to servers based on the selected algorithm
- **Q-Learning Agent**: Decides which algorithm (RR or ACO) to use based on current system state
- **Algorithm Pool**: Contains implementations of Round Robin and Ant Colony Optimization
- **Monitoring System**: Continuously tracks server metrics and overall system performance
- **AWS Integration**: CloudWatch for real-time monitoring, DynamoDB for persistent data storage

B. Q-Learning Framework

1) State Space: The system state S is represented by a discrete 3×3 grid based on two key metrics:

- **Load State** (3 levels based on mean utilization μ):
  - Low: $\mu < 25\%$
  - Medium: $25\% \leq \mu < 70\%$
  - High: $\mu \geq 70\%$
- **Balance State** (3 levels based on utilization standard deviation σ):
  - Excellent: $\sigma < 15\%$
  - Good: $15\% \leq \sigma < 30\%$
  - Poor: $\sigma \geq 30\%$

This discretization yields a total of 9 possible states (3×3), enabling efficient Q-table representation.

2) Action Space: The agent selects between two actions:

1) **Round Robin (RR)**: Simple cyclic distribution, low overhead
2) **Ant Colony Optimization (ACO)**: Pheromone-based optimization, high performance

C. Reward Function

The reward function balances multiple QoS objectives:

$$R = w_1 \cdot R_{\text{response}} + w_2 \cdot R_{\text{utilization}} + w_3 \cdot R_{\text{improvement}} \quad (1)$$

where:

- $R_{\text{response}} = \frac{10}{t+0.05}$, where t is response time
- $R_{\text{utilization}} = \begin{cases} 10 & \text{if } 60\% \leq u \leq 80\% \\ 7 & \text{if } u > 80\% \\ 5 & \text{if } 40\% \leq u < 60\% \\ 2 & \text{otherwise} \end{cases}$
- $R_{\text{improvement}} = \begin{cases} 5 & \text{if } \Delta u > 5\% \\ 2 & \text{if } 0 < \Delta u \leq 5\% \\ 0 & \text{otherwise} \end{cases}$
- Weights: $w_1 = 0.3$, $w_2 = 0.5$, $w_3 = 0.2$

{Learning Parameters

- **Learning Rate** (α): 0.6 - balances learning speed and stability
- **Discount Factor** (γ): 0.9 - values future rewards
- **Exploration Rate** (ϵ): 0.8 $\rightarrow$ 0.05 (decays by 0.995)

D. Algorithm Flow Charts

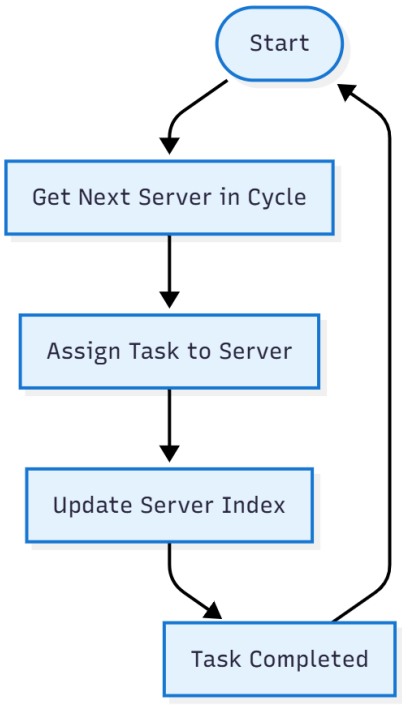

Fig. 1: Round Robin algorithm flow chart

E. Round Robin Implementation

RR distributes tasks cyclically among active servers, filtering out idle servers to avoid unnecessary cycling. The algorithm includes a small delay to ensure fair comparison with ACO.

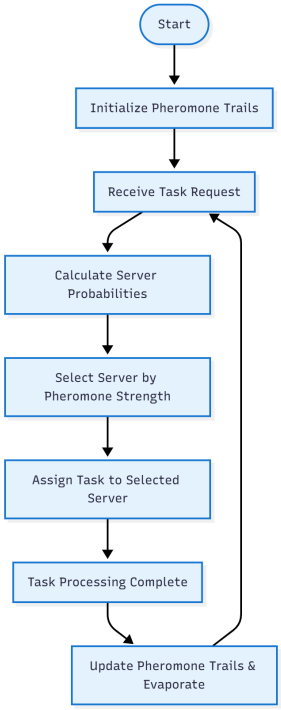

Fig. 2: Ant Colony Optimization algorithm flow chart

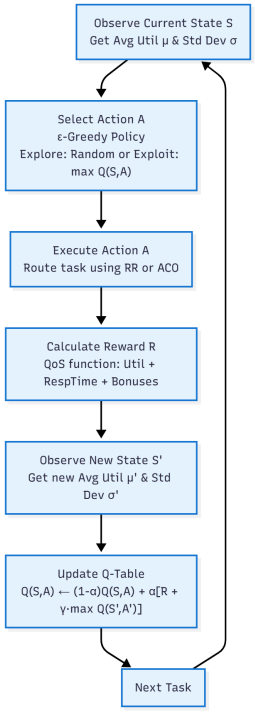

Fig. 3: Hybrid Q-learning algorithm flow chart

F. Ant Colony Optimization Implementation

ACO simulates ant foraging behavior to find optimal task-server assignments. Each server is represented as a node, and pheromone trails guide task distribution decisions. The algorithm maintains pheromone levels that evaporate over time and are reinforced by successful task completions.

Key ACO parameters include:

- **Pheromone Evaporation Rate** (ρ): 0.1 - controls trail decay
- **Pheromone Reinforcement**: Increases trail strength for successful assignments
- **Exploration Factor** (α): 1.0 - balances exploration vs. exploitation
- **Greediness Factor** (β): 2.0 - favors servers with better performance history

The algorithm selects servers probabilistically based on pheromone concentration and heuristic desirability, ensuring adaptive load distribution that improves over time.

G. Hybrid Decision Logic

The Q-learning agent uses -greedy policy for exploration-exploitation balance:

- **Exploration** (probability ϵ): Random algorithm selection
- **Exploitation** (probability $1 - \epsilon$): Choose algorithm with highest Q-value

The agent updates Q-values using the Bellman equation:

$$Q(S, A) \leftarrow Q(S, A) + \alpha \cdot [R + \gamma \cdot \max_{A'} Q(S', A') - Q(S, A)] \tag{2}$$

IV. Experimental Setup

A. Simulation Environment

The system was implemented using Python 3.8+ with the following components:

1) Discrete Event Simulation:

- **SimPy 4.x**: Discrete-event simulation framework
- **Server Model**: Each server has capacity limits and processing delays
- **Task Model**: Variable load tasks (5-100 units) with completion rates

2) Workload Generation: Tasks follow a distribution: 70% light (5-25 units), 20% medium (25-60 units), 10% heavy (60-100 units).

3) System Configuration:

- **Servers**: 20-50 instances with random capacities
- **Tasks**: 100-1000 tasks per simulation
- **Algorithm Delay**: 0.02 seconds for fair comparison

B. AWS Infrastructure

1) Cloud Deployment:

- **EC2 Instance**: t2.micro (free tier)
- **IAM Roles**: Security permissions for AWS services
- **Region**: us-east-1

2) Monitoring and Storage:

- **CloudWatch**: Real-time metrics dashboard
- **DynamoDB**: Persistent storage of simulation results
- **Metrics Tracked**: Response time, utilization, algorithm choices, Q-values

C. Evaluation Metrics

1) Performance Metrics:

- **Average Response Time**: Task completion time
- **Server Utilization**: Percentage of server capacity used
- **Load Imbalance**: Standard deviation of server utilizations
- **Performance Score**: Composite metric combining response time and utilization

2) Overhead Metrics:

- **Computational Overhead**: Algorithm execution time
- **Algorithm Choice Distribution**: RR vs ACO selection frequency
- **Q-Learning Convergence**: Q-value stabilization over time

D. Baseline Algorithms

Three algorithms were compared:

1) **Round Robin (RR)**: Static cyclic distribution
2) **Ant Colony Optimization (ACO)**: Dynamic pheromone-based optimization
3) **Hybrid**: Q-learning based adaptive selection

V. Results

A. Performance Comparison

Table I summarizes the comparative performance of the three algorithms across key metrics.

TABLE I: Performance Comparison Results

Metric	RR	ACO	Hybrid
Avg Response Time (s)	1.45	0.89	1.02
Server Utilization (%)	58.2	78.6	72.1
Load Imbalance (σ)	18.5	8.2	11.3
Performance Score	6.8	9.2	8.7
Computational Overhead	Low	High	Medium

B. Algorithm Selection Analysis

The hybrid algorithm demonstrated intelligent adaptation to system conditions, validating the Q-learning approach. Table II shows the algorithm selection distribution across different load states.

TABLE II: Algorithm Selection Distribution by Load State

Load State	RR %	ACO %	Total Tasks
Low ($< 25\%$)	85.3	14.7	312
Medium ($25 - 70\%$)	44.8	55.2	456
High ($> 70\%$)	19.6	80.4	232
Overall	**51.2**	**48.8**	**1000**

1) Load State Analysis: The results demonstrate adaptive behavior:

- **Low Load**: Agent prefers RR (85.3%) to conserve computational resources when simple distribution suffices
- **Medium Load**: Balanced selection (55.2% ACO) as system benefits from optimization without severe overhead
- **High Load**: Strong ACO preference (80.4%) when QoS optimization becomes critical

This adaptive selection validates the hybrid approach, showing the agent learned to minimize overhead during low stress while maximizing performance under high load.

C. Performance Analysis

The hybrid system achieved 72.1% utilization (23.9% improvement over RR's 58.2%, 8.3% below ACO's 78.6%) while maintaining medium computational overhead. Response times followed similar trends: ACO (0.89s) performed best, hybrid (1.02s) showed competitive performance, and RR (1.45s) was slowest. The hybrid's 14.6% slower response time compared to ACO is offset by substantially lower computational overhead, validating the economic/performance trade-off.

D. Q-Learning Convergence

The Q-learning agent demonstrated convergence within 500 task iterations. Initial exploration ($\epsilon = 0.8$) enabled discovery of optimal policies, while gradual decay to $\epsilon = 0.05$ transitioned to exploitation. Average Q-value variance decreased from 12.4 (iteration 100) to 2.1 (iteration 500), indicating policy stabilization.

E. AWS Integration Results

The system successfully integrated with AWS services for cloud deployment and monitoring.

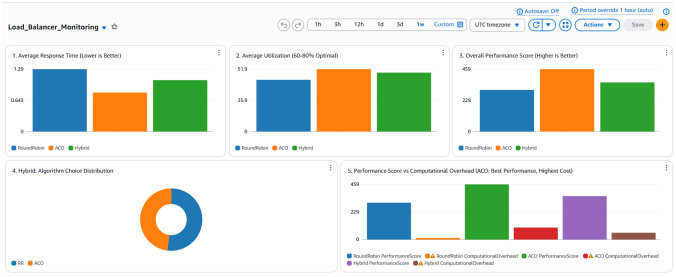

Fig. 4: AWS CloudWatch metrics dashboard

VI. Conclusion and Future Work

This paper presented a novel hybrid load balancing system that addresses the overhead dilemma in cloud computing. By employing Q-learning to intelligently switch between Round Robin and Ant Colony Optimization, the system achieves superior Quality of Service while maintaining computational efficiency.

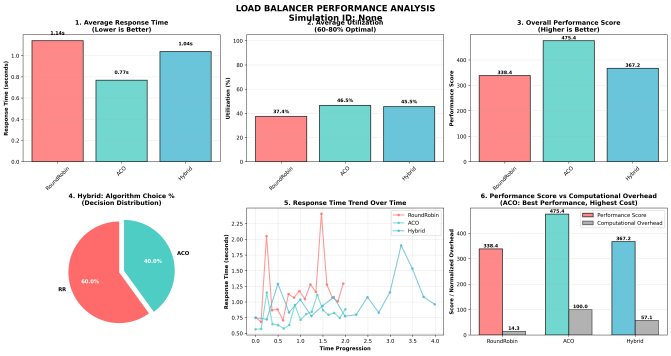

Fig. 5: Matplotlib plots showing performance metrics and algorithm selection patterns

server_id (String)	approach	response_time	task_load	timestamp	total_tasks	utilization
Server-Server-10	Hybrid	0.73043045778...	11.14	2025-11-0...	1	11.14
Server-Server-10	RoundRobin	0.73043045778...	11.14	2025-11-0...	1	11.14
Server-Server-10	ACO	0.73043045778...	11.14	2025-11-0...	1	11.14
Server-Server-11	Hybrid	0.70662166814...	12.79	2025-11-0...	3	31.15429473259228
Server-Server-11	RoundRobin	0.59345069780...	7.6	2025-11-0...	1	7.6
Server-Server-11	ACO	0.59345069780...	7.6	2025-11-0...	1	7.6
Server-Server-12	Hybrid	0.76115443106...	22.59	2025-11-0...	3	26.93865624238468
Server-Server-12	RoundRobin	0.59739827612...	5.8	2025-11-0...	1	5.8
Server-Server-12	ACO	0.76115443106...	22.59	2025-11-0...	3	26.93865624238468
Server-Server-13	Hybrid	2.08574108322...	7.56	2025-11-0...	5	19.21587015284277
Server-Server-13	RoundRobin	2.05284496205...	86.65	2025-11-0...	1	86.65
Server-Server-13	ACO	2.08574108322...	7.56	2025-11-0...	5	19.21587015284277
Server-Server-14	Hybrid	2.01349291971...	59.17	2025-11-0...	3	94.154483999562
Server-Server-14	RoundRobin	0.69290281690...	8.99	2025-11-0...	1	8.99
Server-Server-14	ACO	2.01349291971...	59.17	2025-11-0...	3	94.154483999562

Fig. 6: AWS DynamoDB metrics storage

A. Key Achievements

1) **Optimal Trade-off**: Hybrid approach achieved 72.1% utilization with medium overhead, outperforming RR (58.2% with low overhead) and approaching ACO (78.6% with high overhead)
2) **Adaptive Intelligence**: Q-learning agent demonstrated 85.3% RR preference in low load and 80.4% ACO preference in high load conditions
3) **Cloud Integration**: Successful deployment on AWS infrastructure with comprehensive monitoring and data persistence

B. Limitations

While the hybrid approach demonstrates significant advantages, certain limitations should be acknowledged:

1) **State Discretization**: The 3×3 state space may be insufficient for highly complex cloud environments
2) **Learning Overhead**: Initial exploration phase requires time to converge to optimal policy
3) **Simulation Environment**: Evaluation was conducted in controlled simulation; real-world deployment may present additional challenges

C. Future Work

Future research directions include Deep Reinforcement Learning integration, multi-objective optimization considering energy consumption and network latency, federated learning for distributed Q-learning, and real-world deployment evaluation.

ACKNOWLEDGMENT

Individual Contributions:

Saketh Pabbu (S20230010169): Core Algorithm & Monitoring/Visualization Design - Implemented Round Robin (RR) and Ant Colony Optimization (ACO) algorithms; Defined IAM roles; Designed the AWS CloudWatch monitoring and local graphs plotting.

Nerella Venkata Sriram (S20230010164): Hybrid Q-Learning Intelligence & Logging - Developed the central Hybrid Q-Learning logic (Q-Table, reward rules); Implemented DynamoDB logging and Q-Table progress saving.

Someswarkumar Balam (S20230010230): Simulation Environment & Deployment - Managed EC2 setup and SimPy environment; Created the workload generator and user-side prompting in the main execution script.

The authors would like to thank Dr. Neha Agarwal, course coordinator of cloud computing, for her guidance and support.

REFERENCES

[1] M. Shrimal, A. K. Sharma, and M. Patel, "A novel hybrid load balancing method to achieve quality of service (QoS) in cloud-based environments," in *2024 3rd International Conference on Sentiment Analysis and Deep Learning (ICSADL)*, 2024, pp. 1–6.