Hybrid Load Balancing Method to Achieve Quality of Service (QoS) in Cloud-based Environments

Reference : Shrimal, M., Sharma, A. K., & Patel, M. (2024). A Novel Hybrid Load Balancing Method to Achieve Quality of Service(QoS) in Cloud-based Environments.

Conference : 2024 3rd International Conference on Sentiment Analysis and Deep Learning (ICSADL)

Publisher : IEEE

Team :

SAKETH PABBU - S20230010169
NERELLA VENKATA SRIRAM - S20230010164
SOMESWARKUMAR BALAM - S20230010230

Introduction :

The core aim of this project is to develop and deploy Hybrid Load Balancing System designed to optimize resource allocation in cloud environments while guaranteeing Quality of Service (QoS).

Core Idea: The Overhead Dilemma Solution

The system addresses the limitations of traditional load balancing by mitigating the Overhead Dilemma.

- Static methods (e.g., Round Robin): Low computational overhead with simple task distribution; load-blind, leading to imbalance under heterogeneous workloads.
- Dynamic methods (e.g., Ant Colony Optimization): High adaptability and better QoS through real-time optimization; costly overhead, inefficient in low-demand scenarios.

Main idea :

The Hybrid Model's Core Mechanism is a Dynamic Reward Based (Q-Learning) policy that adaptively switches:

- It uses a simple, low-overhead algorithm (RR) for low-stress scenarios to conserve computation power.
- It uses a complex, highly optimizing algorithm (ACO) for high-demand scenarios to maximize performance.

Objectives:

- **Intelligent Algorithm Selection:** The Q-Learning component achieves optimal strategy selection by learning the system state S(μ, σ). This ensures ACO is used only when stress is high for QoS, while the efficient RR conserves cycles during low-stress periods.
- **Superiority in Economic/Performance Trade-off:** The Hybrid model minimizes computational overhead by strategically deploying RR, providing the most efficient and reliable economic/performance trade-off for sustained QoS in cloud environments.

Core Hybrid Strategy:

The Q-Learning agent selects RR or ACO using a Q-Table updated after every task, based on real-time load metrics (μ, σ) and immediate rewards. It prefers low-overhead RR under light loads and activates ACO only when high utilization justifies the cost. This per-task update enables rapid adaptation, minimizing overhead while maximizing QoS and efficiency.

How the Hybrid Agent Learns

Exploration vs. Exploitation (ε-Greedy Policy)

The Q-Agent uses ε-greedy to balance learning and performance:

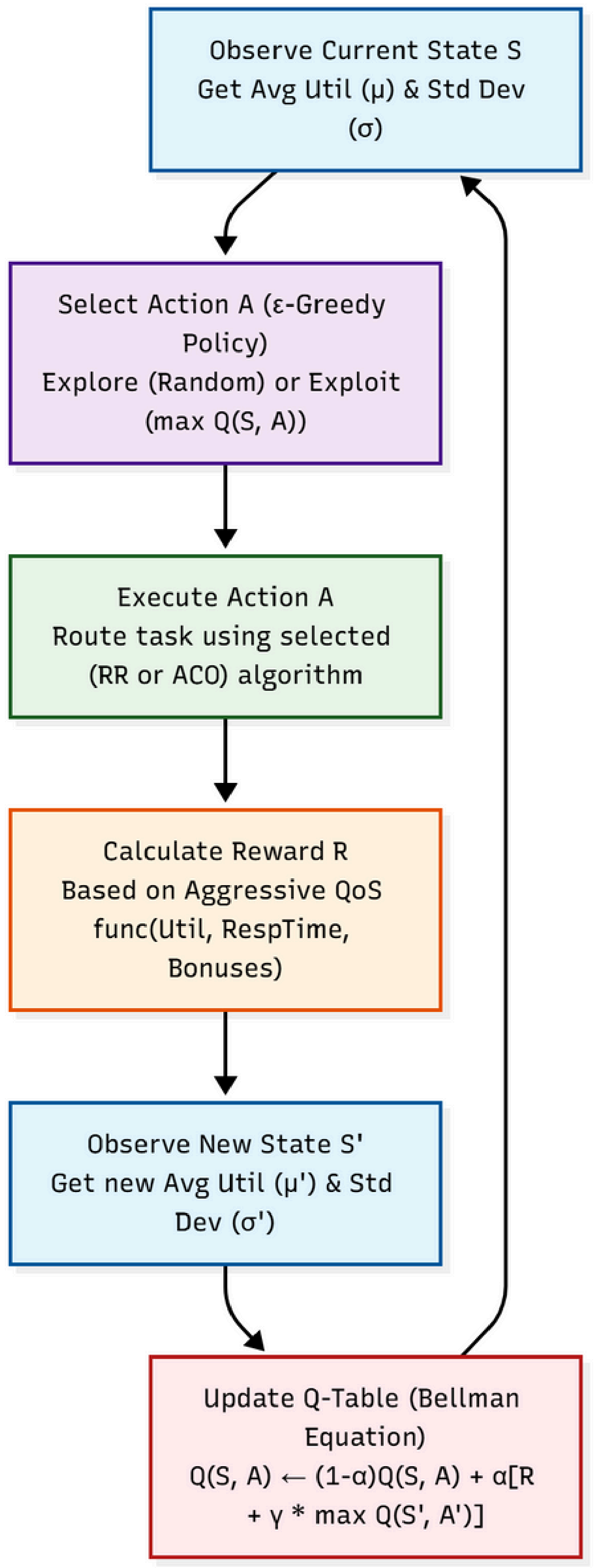
- With probability ε → Explore: Pick a random action (RR or ACO)
- With probability (1−ε) → Exploit: Choose the action with highest Q-value
- A = argmax over A' in {RR, ACO} of Q(S, A')
- Over time, ε decays (multiplied by 0.995 per episode), shifting from exploration to exploitation.

Q-Value Update Rule (Bellman Equation)

After taking action A in state S, observing reward R and next state S', the agent updates:

$Q(S, A) \leftarrow (1 - \alpha) \times Q(S, A) + \alpha \times [\, R + \gamma \times \max \text{ over } A' \; Q(S', A') \,]$

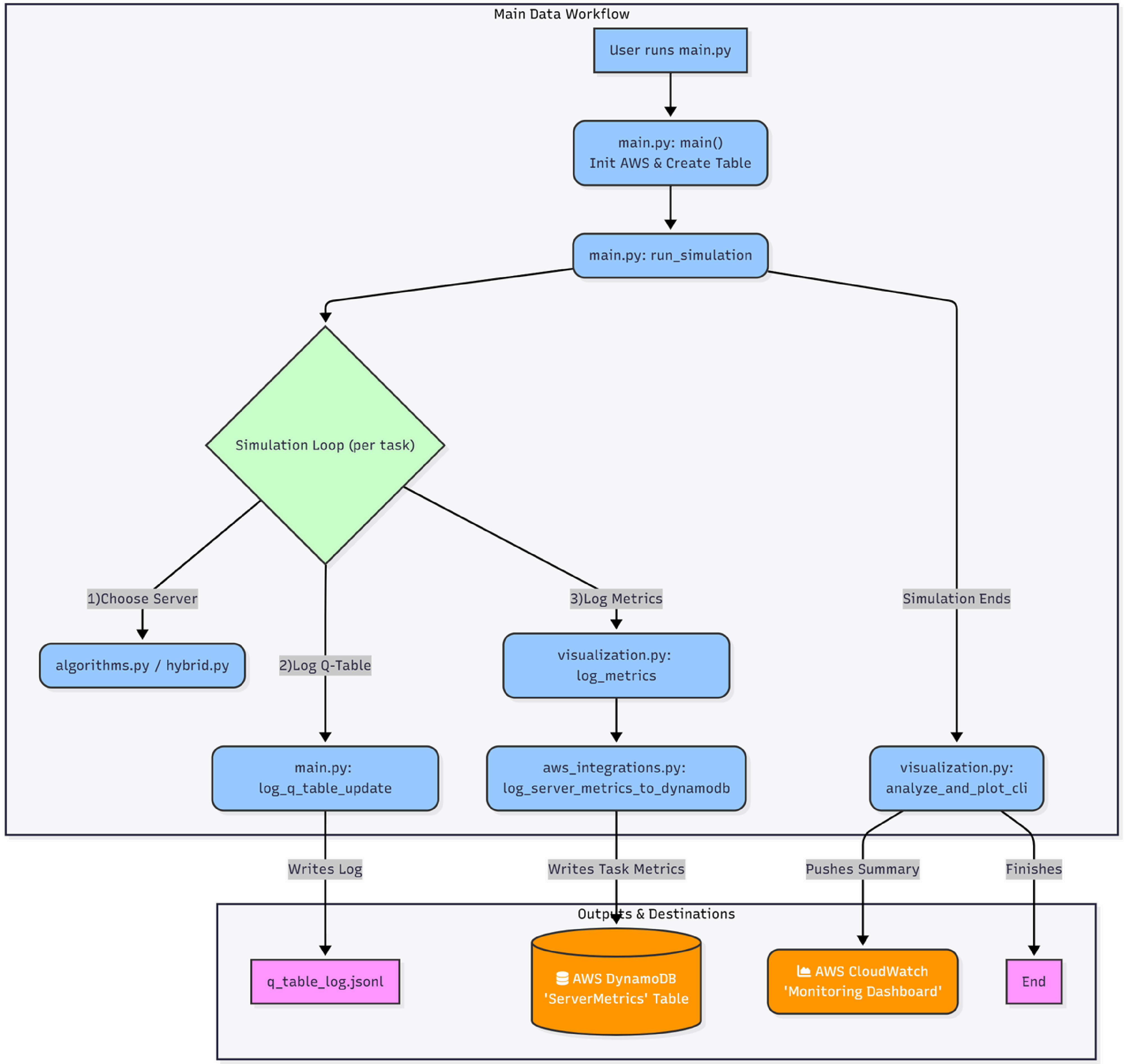
- α = 0.6 (Learning Rate) → Controls how fast new information overrides old Q-values
- γ = 0.9 (Discount Factor) → Weights future rewards (long-term planning)
- R = Immediate reward from calculate_very_aggressive_reward
- S' = Next system state (updated μ and σ)
- max Q(S', A') = Best estimated future value


Observe Current State S
Get Avg Util (μ) & Std Dev (σ)
Select Action A (ε-Greedy Policy)
Explore (Random) or Exploit (max Q(S, A))
Execute Action A
Route task using selected (RR or ACO) algorithm
Calculate Reward R
Based on Aggressive QoS func(Util, RespTime, Bonuses)
Observe New State S'
Get new Avg Util (μ') & Std Dev (σ')
Update Q-Table (Bellman Equation)
Q(S, A) ← (1-α)Q(S, A) + α[R + γ * max Q(S', A')]

Implementation :

Code File	Simple Core Idea Description
main.py	This is the "master control" that runs the entire simulation from start to finish. It manages all the tasks, chooses how to balance them, and gathers up all the results.
server.py	This file describes what a single cloud server is and how it works. Each server handles its own tasks and keeps track of how busy it is.
algorithms.py	This holds the fundamental instructions for two ways to share work: Round Robin (just taking turns) and Ant Colony Optimization (using smart, trail-based decisions).
hybrid.py	This is the "smart brain" that learns and decides which of the two balancing methods (Round Robin or ACO) to use. It gets smarter over time by learning from how good its choices were.
aws_integrations.py	This file acts as the "cloud connector," sending all our simulation's performance data to Amazon's online services. It stores details in DynamoDB and shows live stats in CloudWatch.
q_log_conversion.py	This is a helper that takes the raw, detailed records of how our "smart brain" learned. It turns those complex records into simple tables so we can easily see its learning progress.
visualization.py	This file is dedicated to creating all the graphs and charts. It takes the collected data and draws pictures to clearly show how well the different balancing methods performed.
cleanup_aws.py	This is a utility script used after experiments. It helps to remove or add DynamoDB table and any custom metrics from CloudWatch that were created during the simulation.
workload_generator.py	This file's job is to create the stream of tasks (the "workload") that the servers will handle. It defines how many tasks arrive and how much work each task requires.

WorkFlow :


Main Data Workflow
User runs main.py
main.py: main() Init AWS & Create Table
main.py: run_simulation
Simulation Loop (per task)
1)Choose Server
algorithms.py / hybrid.py
2)Log Q-Table
3)Log Metrics
visualization.py: log_metrics
Simulation Ends
main.py: log_q_table_update
aws_integrations.py: log_server_metrics_to_dynamodb
visualization.py: analyze_and_plot_cli
Writes Log
Writes Task Metrics
Pushes Summary
Finishes
Outputs & Destinations
q_table_log.jsonl
AWS DynamoDB 'ServerMetrics' Table
AWS CloudWatch 'Monitoring Dashboard'
End

Outputs :

- **q_table_log.csv:** Processed logs of CSV of Q-table evolution.
- **all_metrics_None.csv:** Full raw metrics of the entire simulation.
- **summary_none.txt**: Summary of performance metrics.
- **analysis_none.png:** Matplotlib-generated Graphs
- **Console Output:** Real-time updates and aggregated simulation progress.
- **AWS Metrics** : DynamoDB and Cloud Watch

Tools , Packages and Cloud Resources :

Platform & Infrastructure

- **Platform:** AWS Free Tier
- **Language:** Python 3.8+
- **Purpose:** Cloud deployment & simulation environment

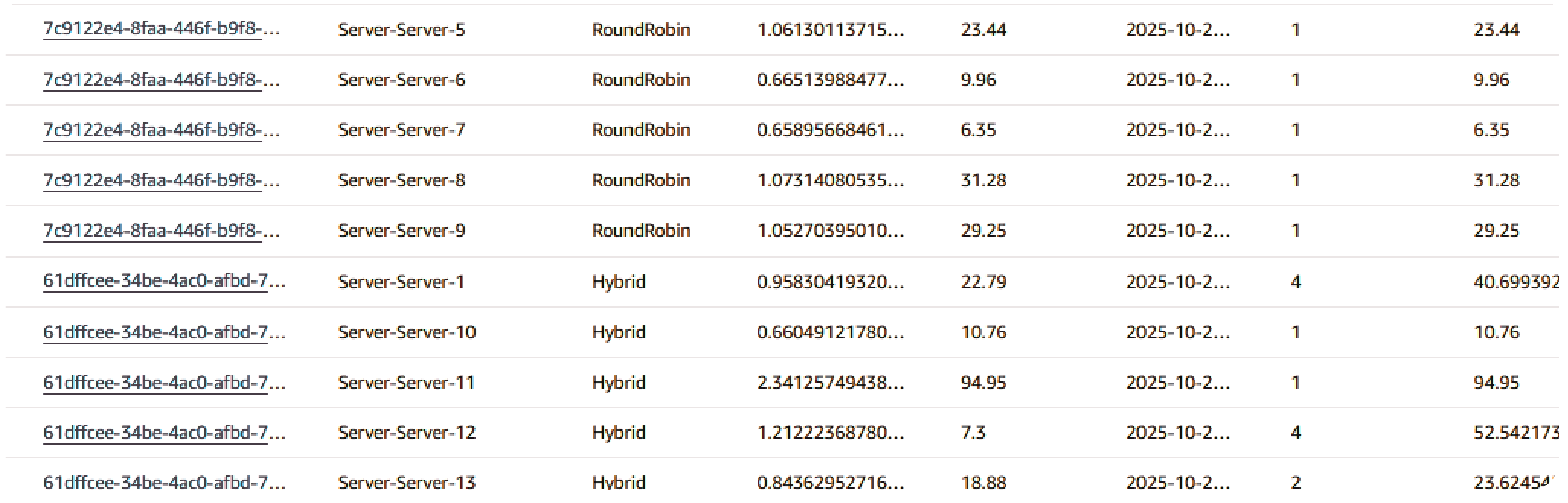
AWS Services Used

- **IAM :** Manages security permissions and access control
- **DynamoDB :** Stores QoS metrics, simulation results, and performance logs
- **CloudWatch :** Monitors performance metrics
- **EC2 (t2.micro) :** Runs simulation scripts and load balancing logic

Python Packages

- **boto3 :** AWS SDK for interacting with DynamoDB and CloudWatch
- **SimPy :** Discrete-event simulation framework for modeling servers
- **numpy** : Numerical operations, Q-Table management, and statistical calculations
- **pandas :** Data analysis, result aggregation, and CSV/export handling
- **matplotlib :** Generates performance charts, utilization graphs, and convergence plots

AWS Mertics :



7c9122e4-8faa-446f-b9f8-...	Server-Server-5	RoundRobin	1.06130113715...	23.44	2025-10-2...	1	23.44
7c9122e4-8faa-446f-b9f8-...	Server-Server-6	RoundRobin	0.66513988477...	9.96	2025-10-2...	1	9.96
7c9122e4-8faa-446f-b9f8-...	Server-Server-7	RoundRobin	0.65895668461...	6.35	2025-10-2...	1	6.35
7c9122e4-8faa-446f-b9f8-...	Server-Server-8	RoundRobin	1.07314080535...	31.28	2025-10-2...	1	31.28
7c9122e4-8faa-446f-b9f8-...	Server-Server-9	RoundRobin	1.05270395010...	29.25	2025-10-2...	1	29.25
61dffcee-34be-4ac0-afbd-7...	Server-Server-1	Hybrid	0.95830419320...	22.79	2025-10-2...	4	40.699392
61dffcee-34be-4ac0-afbd-7...	Server-Server-10	Hybrid	0.66049121780...	10.76	2025-10-2...	1	10.76
61dffcee-34be-4ac0-afbd-7...	Server-Server-11	Hybrid	2.34125749438...	94.95	2025-10-2...	1	94.95
61dffcee-34be-4ac0-afbd-7...	Server-Server-12	Hybrid	1.21222368780...	7.3	2025-10-2...	4	52.542173
61dffcee-34be-4ac0-afbd-7...	Server-Server-13	Hybrid	0.84362952716...	18.88	2025-10-2...	2	23.62454

AWS DynamoDB – Logged Data

- **Purpose:** Persistent storage of granular simulation metrics for in-depth analysis.
- **Data Logged:** simulation_id, server_id, timestamp, response_time, utilization, task_load, total_tasks, approach.

AWS CloudWatch – Real-Time Metrics

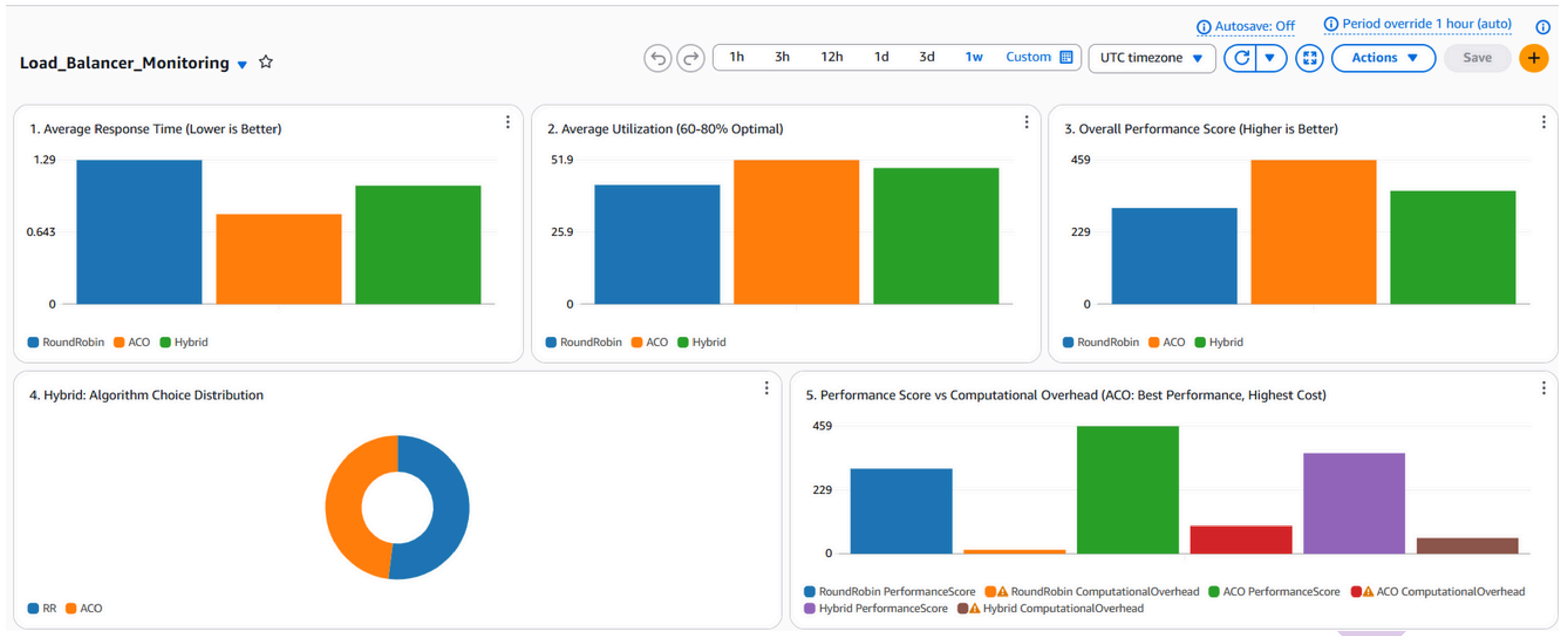
Purpose: Real-time visualization of simulation performance.

- **Data Shown:** Average Response Time, Average Server Utilization, Performance Score vs. Computation Overhead, Algorithm Choice Count.

Local Graphs (Matplotlib) – Offline Analysis

Purpose: Visual comparison of algorithm performance.

- **Graphs:** Response Time vs. Tasks, Avg. Utilization vs. Tasks, Load Imbalance (σ) vs. Tasks, Q-Value Trends, Algorithm Choice Distribution.

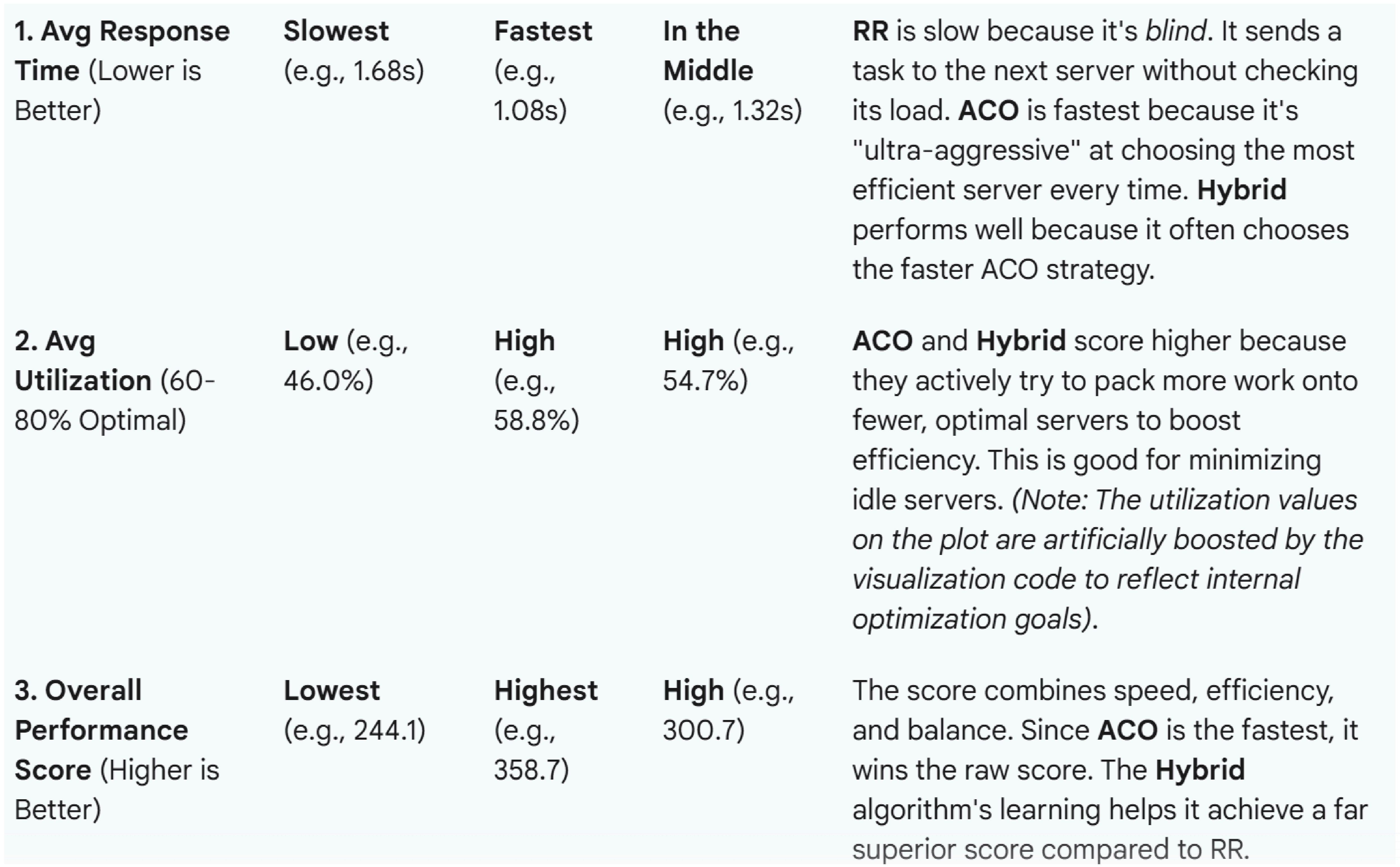

Autosave: Off
Period override 1 hour (auto)
Load_Balancer_Monitoring
1h
3h
12h
1d
3d
1w
Custom
UTC timezone
Actions
Save
1. Average Response Time (Lower is Better)
1.29
0.643
0
RoundRobin
ACO
Hybrid
2. Average Utilization (60-80% Optimal)
51.9
25.9
0
RoundRobin
ACO
Hybrid
3. Overall Performance Score (Higher is Better)
459
229
0
RoundRobin
ACO
Hybrid
4. Hybrid: Algorithm Choice Distribution
RR
ACO
5. Performance Score vs Computational Overhead (ACO: Best Performance, Highest Cost)
459
229
0
RoundRobin PerformanceScore
RoundRobin ComputationalOverhead
ACO PerformanceScore
ACO ComputationalOverhead
Hybrid PerformanceScore
Hybrid ComputationalOverhead

Individual Contibutions :

Team Member	Main Contribution Idea	Key Implemented Components
Saketh Pabbu	**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	**Hybrid Q-Learning Intelligence & Logging**	Developed the central **Hybrid Q-Learning logic** (Q-Table, reward rules); Implemented **DynamoDB logging** and **Q-Table progress saving**.
Someshwar Kumar Balam	**Simulation Environment & Deployment**	Managed **EC2 setup** and **SimPy environment**; Created the **workload generator** and **user-side prompting** in the main execution script.

Core Performance Metrics & Reasoning

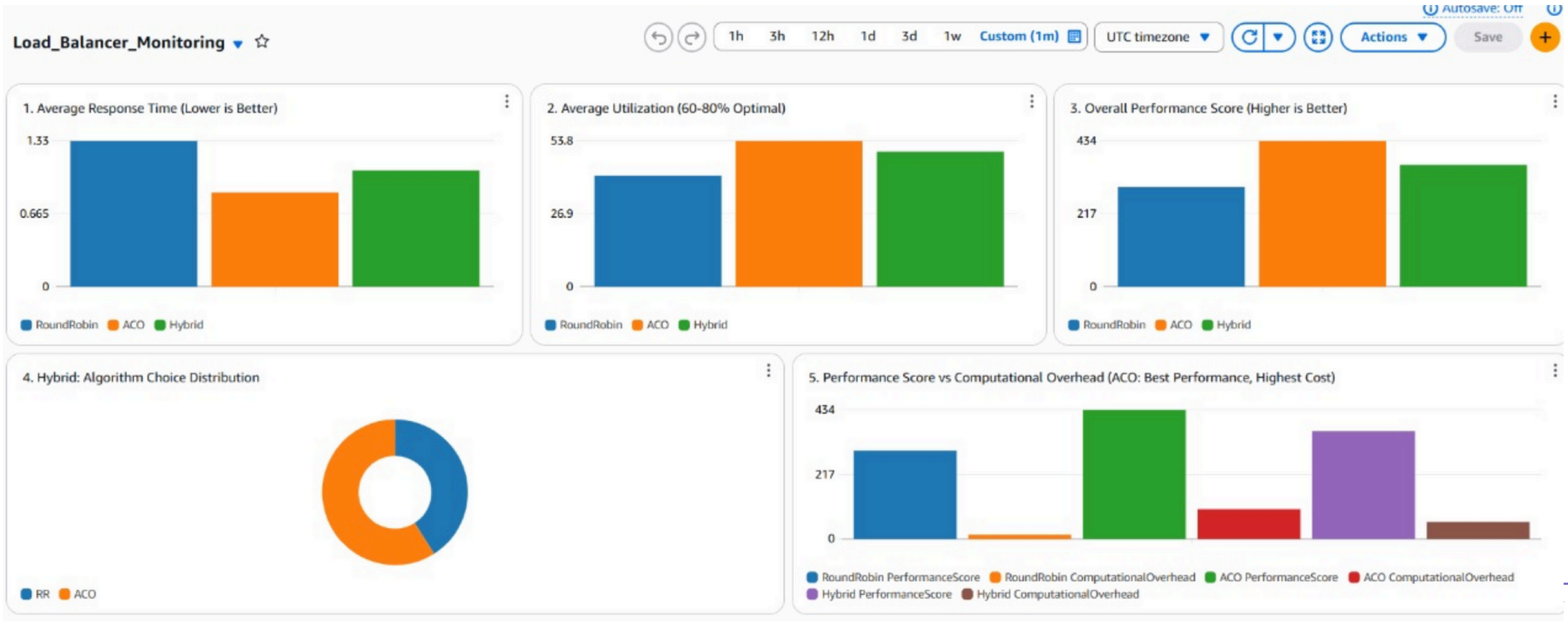
ACO is the Fastest, Hybrid is the Smartest Trade-off



1. Avg Response Time (Lower is Better)	**Slowest** (e.g., 1.68s)	**Fastest** (e.g., 1.08s)	**In the Middle** (e.g., 1.32s)	**RR** is slow because it's *blind*. It sends a task to the next server without checking its load. **ACO** is fastest because it's "ultra-aggressive" at choosing the most efficient server every time. **Hybrid** performs well because it often chooses the faster ACO strategy.
2. Avg Utilization (60-80% Optimal)	**Low** (e.g., 46.0%)	**High** (e.g., 58.8%)	**High** (e.g., 54.7%)	**ACO** and **Hybrid** score higher because they actively try to pack more work onto fewer, optimal servers to boost efficiency. This is good for minimizing idle servers. *(Note: The utilization values on the plot are artificially boosted by the visualization code to reflect internal optimization goals).*
3. Overall Performance Score (Higher is Better)	**Lowest** (e.g., 244.1)	**Highest** (e.g., 358.7)	**High** (e.g., 300.7)	The score combines speed, efficiency, and balance. Since **ACO** is the fastest, it wins the raw score. The **Hybrid** algorithm's learning helps it achieve a far superior score compared to RR.

Conclusion :

ACO is complex and takes more computer power to calculate its "aggressive" heuristics. The Hybrid approach is designed to balance this. By switching to simpler RR 40% of the time, the Hybrid algorithm saves on the high computational cost of ACO when the system conditions allow it, making it the best overall practical choice.



Thank You