Performance Benchmarking and Intelligent Instance Selection for AWS EC2

Student Name: Longjie Su **Student ID:** 225040158

1. Introduction and Motivation

Cloud infrastructure selection involves complex trade-offs between performance and cost. With hundreds of EC2 instance types available, users often struggle to identify the optimal configuration for their specific workloads. Poor selection results in either performance bottlenecks (under-provisioning) or unnecessary expenses (over-provisioning).

The primary motivation of this project is to demystify these trade-offs through empirical analysis. We aim to answer: How do General Purpose (m5), Compute Optimized (c5), and Burstable (t3) instances differ under real-world application loads? Furthermore, can selection process be automated?

This project contributes:

1. Comprehensive Benchmarking: Evaluating CPU, I/O, Network, Web (Nginx), and Database (MySQL) performance.
2. Cost-Efficiency Analysis: Quantifying the "performance per dollar" for each instance.
3. Intelligent Recommendation: A Machine Learning-based decision support system that recommends instances based on workload characteristics.

2. Methodology and Technical Approach

2.1 Experimental Environment

Three representative EC2 instances were selected in the us-east-1 region (Ubuntu 24.04 LTS):

- t3.medium: Burstable, cost-effective (2 vCPU, 4GB RAM).
- m5.large: General purpose, consistent performance (2 vCPU, 8GB RAM).
- c5.large: Compute optimized (2 vCPU, 4GB RAM).

2.2 Benchmarking Suite

A layered testing approach using industry-standard open-source tools was employed:

1. Synthetic Layer: Sysbench (CPU/Memory), FIO (Disk I/O), and iperf3 (Network TCP throughput) to measure raw hardware capabilities .
2. Application Layer:

- Nginx Web Server: Tested using wrk to measure Requests Per Second (RPS) and latency under varying concurrency (100-1000 connections) and payload sizes (1KB, 10KB, 100KB).
- MySQL Database: Tested using sysbench oltp_read_write to measure Transactions Per Second (TPS) and Query Per Second (QPS) across varying thread counts (1-8).

3. Analysis Layer: Automated Python pipelines utilizing pandas for data cleaning, matplotlib/seaborn for visualization, and scikit-learn for the recommendation model.

3. Evaluation and Results

3.1 Synthetic Benchmark Summary

Preliminary results indicated that c5.large dominates in raw CPU throughput (701 events/s vs. ~640 for others) and memory bandwidth. t3.medium demonstrated impressive network burst capabilities (39.4 Gbps peak) but suffered significant degradation (-33%) under parallel load, confirming its "burstable" nature.

3.2 Real-World Application: MySQL Performance

Database performance is critical for most cloud applications. The instances were evaluated under Read/Write intensive workloads.

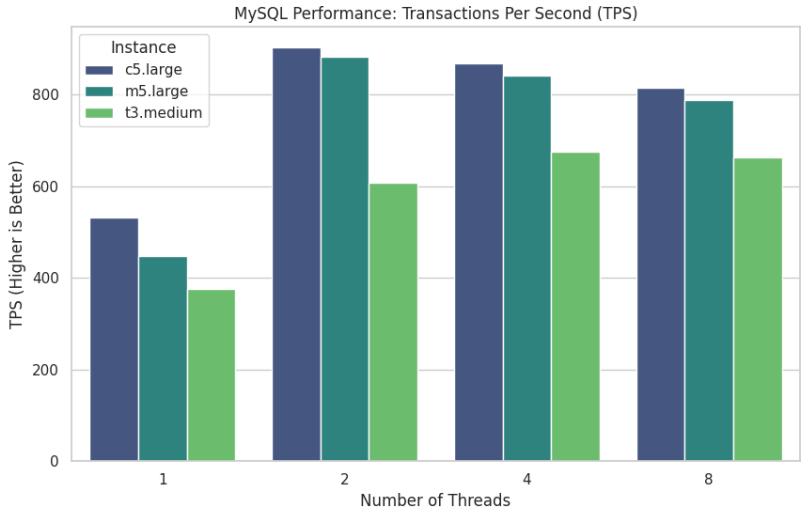

Figure 1: MySQL Transactions Per Second (TPS) Comparison.

As shown in Figure 1, c5.large consistently outperforms other instances. At 8 threads, c5.large maintains ~810 TPS, while t3.medium drops to ~660 TPS. This 20%+ performance gap highlights the impact of the c5's superior processor clock speed on database transaction handling.

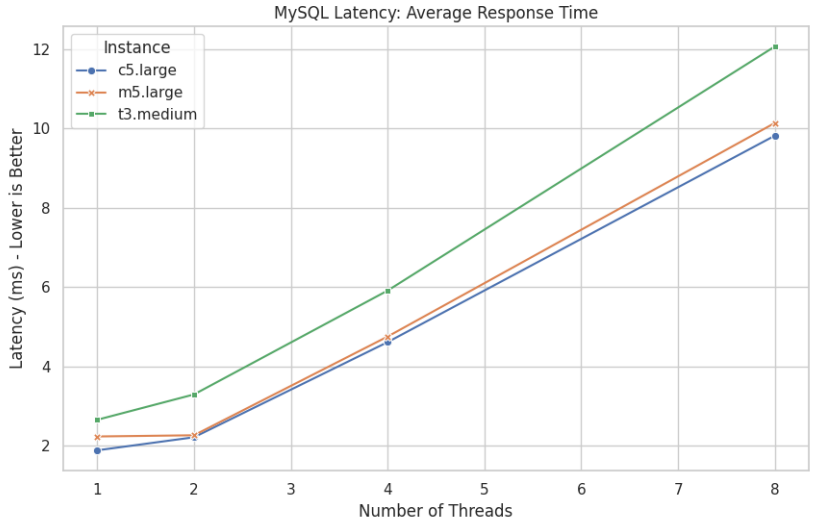

Figure 2: MySQL Average Latency vs. Threads.

Latency analysis (Figure 2) reveals that t3.medium experiences the steepest latency degradation as concurrency increases, reaching over 12ms at 8 threads compared to ~9.8ms for c5.large. m5.large provides a middle ground, benefiting from its larger memory (8GB) for buffer pools, but still trailing c5.large in raw processing speed.

3.3 Real-World Application: Nginx Web Server

Web server performance was tested to simulate high-concurrency traffic.

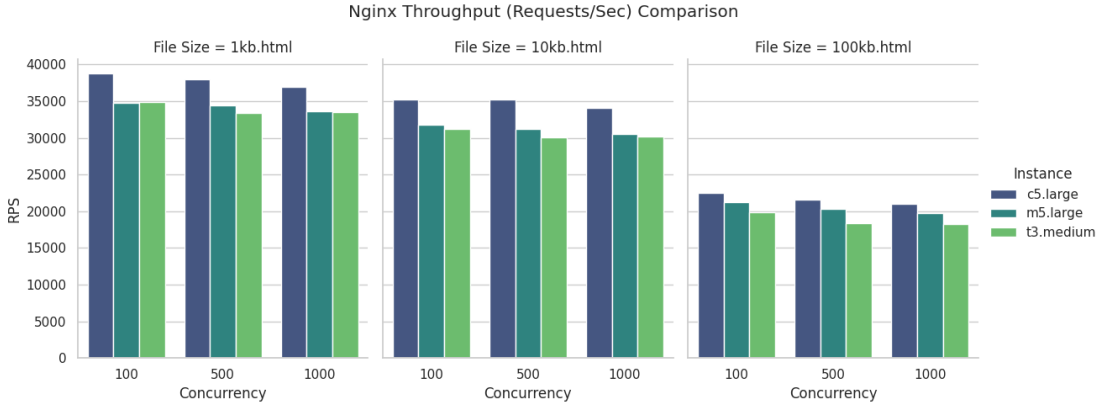

Figure 3: Nginx Throughput (RPS) across different file sizes.

For small static files (1KB), c5.large achieves the highest throughput, peaking at over 38,000 RPS. However, for larger files (100KB), the bottleneck shifts from CPU to Network Bandwidth. Here, the instances perform similarly due to the network bandwidth limits of the "Large" instance class, proving that CPU optimization yields diminishing returns for bandwidth-bound tasks.

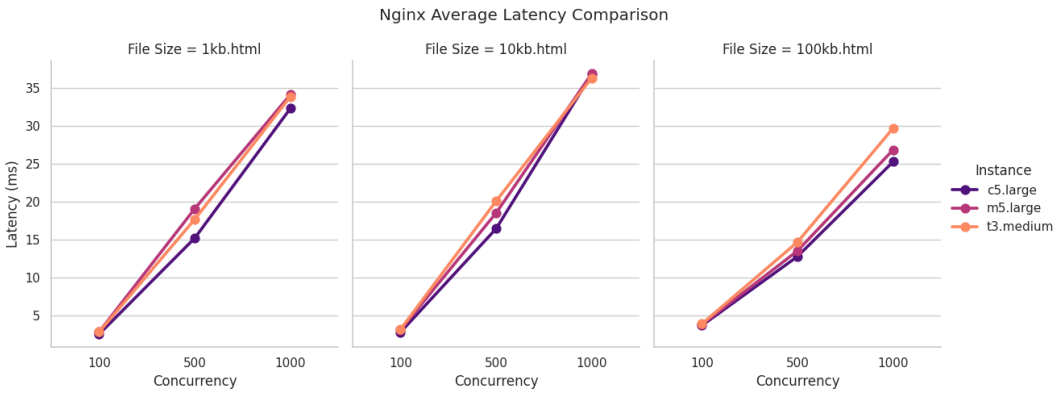

Figure 4: Nginx Latency Comparison.

Figure 4 demonstrates that all instances maintain low latency (<5ms) at low concurrency. However, at 1000 concurrent connections, latency spikes linearly. c5.large maintains the lowest latency curve, validating its suitability for latency-sensitive web applications.

3.4 Cost-Efficiency Analysis

High performance does not always equal high value. "Requests Served Per Dollar" was calculated based on on-demand pricing (t3.medium: $0.0416/hr, c5.large: $0.085/hr).

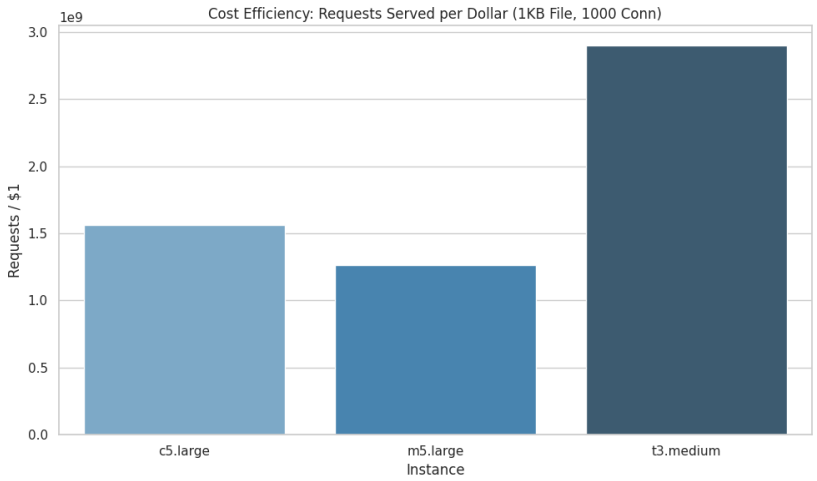

Figure 5: Cost Efficiency (Nginx 1KB File, 1000 Connections).

Surprisingly, t3.medium is the undisputed winner in cost efficiency. As shown in Figure 5, it delivers nearly 2x the requests per dollar compared to c5.large. For non-critical or development workloads where peak performance is less important than budget, t3.medium is the optimal choice.

4. Machine Learning Instance Recommender

To operationalize our findings, an ML-based "Instance Recommender" was built using a

Decision Tree Classifier.

4.1 Model Design

The model was trained on a knowledge base derived from our benchmark data.

- Features: CPU Intensity (1-10), Memory (GB), Workload Type (Web/DB/Dev), Budget Sensitivity (Yes/No).
- Target: Instance Type (t3, m5, c5).

4.2 Decision Logic and Results

The trained model revealed clear decision boundaries consistent with our empirical data.

- Rule 1: If CPU demand is high (>7.0), select c5.large.
- Rule 2: If Memory demand is high (>5.5GB), select m5.large.
- Rule 3: For low resource demand or budget-sensitive projects, select t3.medium .

Evaluation Scenarios: The model achieved 100% confidence in test scenarios:

1. High-Traffic Web Server: Correctly recommended c5.large.
2. Production MySQL DB: Correctly recommended m5.large (prioritizing memory).
3. Student Dev Environment: Correctly recommended t3.medium.

5. Discussion and Conclusion

This project successfully characterized the performance distinctiveness of AWS EC2 instances.

1. c5.large is the "Performance King," ideal for computation-heavy and high-concurrency web workloads.
2. m5.large is the "Balanced Choice," specifically for database workloads requiring larger memory footprints.
3. t3.medium is the "Value King," offering the best cost-efficiency despite lower stability under sustained load.

Future work would involve integrating Spot Instance pricing into the recommender system and focusing continuously on a long-term stability test (12+ hours) to observe t3 credit depletion was not conducted.

Appendix

Artifact Description

The complete codebase is available in the GitHub repository, structured for reproducibility.

- **Repository Structure:**
    - /scripts/: Shell scripts for environment setup (setup_environment.sh) and benchmarking (mysql_benchmark.sh, nginx_benchmark.sh).
    - /results/: Raw JSON data from all experiments.
    - /analysis/: Python scripts (analyze_results.py, ml_recommender.py) for generating charts and the recommendation model.
    - requirements.txt: Python dependencies.