CLOUD COMPUTING IA-2 PRESENTATION

Differentiated Quality of Experience Scheduling for Deep Learning Inference With Docker Containers in the Cloud

Presented By: Hridayansh Bajani
Roll No.: 16010122010

Under the Guidance of: Professor Zaheed Sheikh

ABSTRACT SUMMARY

- With the rapid growth of deep learning applications, services now heavily rely on backend cloud infrastructure to deliver fast responses.
- These services often operate under tight budget constraints and require different levels of Quality of Experience (QoE).
- Existing cloud providers do not support QoE-based resource scheduling.
- This paper proposes DQoES, a novel scheduling system that accepts user-defined QoE targets and dynamically adjusts resources to meet those targets.
- Extensive experiments show that DQoES can deliver up to 8x more satisfied models compared to traditional scheduling systems.

INTRODUCTION

- The rise of big-data-driven services such as Apple FaceID and Google AdSense has brought deep learning to the forefront.
- Deploying these models in the cloud is resource-intensive and costly.
- Users expect low latency experiences (e.g., face unlock), but not all applications require the same responsiveness.
- The challenge is: How do we balance QoE with the cost of using limited cloud resources?
- Current solutions provide resource-level configuration, not QoE-level customization.

PROBLEM STATEMENT

- **Cloud users face a dilemma:**
  - Provide better QoE → More cost.
  - Save cost → Poor user experience.
- **Applications have diverse requirements (real-time vs tolerable delays).**
- **Existing schedulers do not:**
  - Accept QoE as input.
  - Dynamically adjust resources based on user experience.
- **Need for a system that:**
  - Understands QoE targets.
  - Adjusts resources at runtime.
  - Works in multi-tenant environments like Docker Swarm.

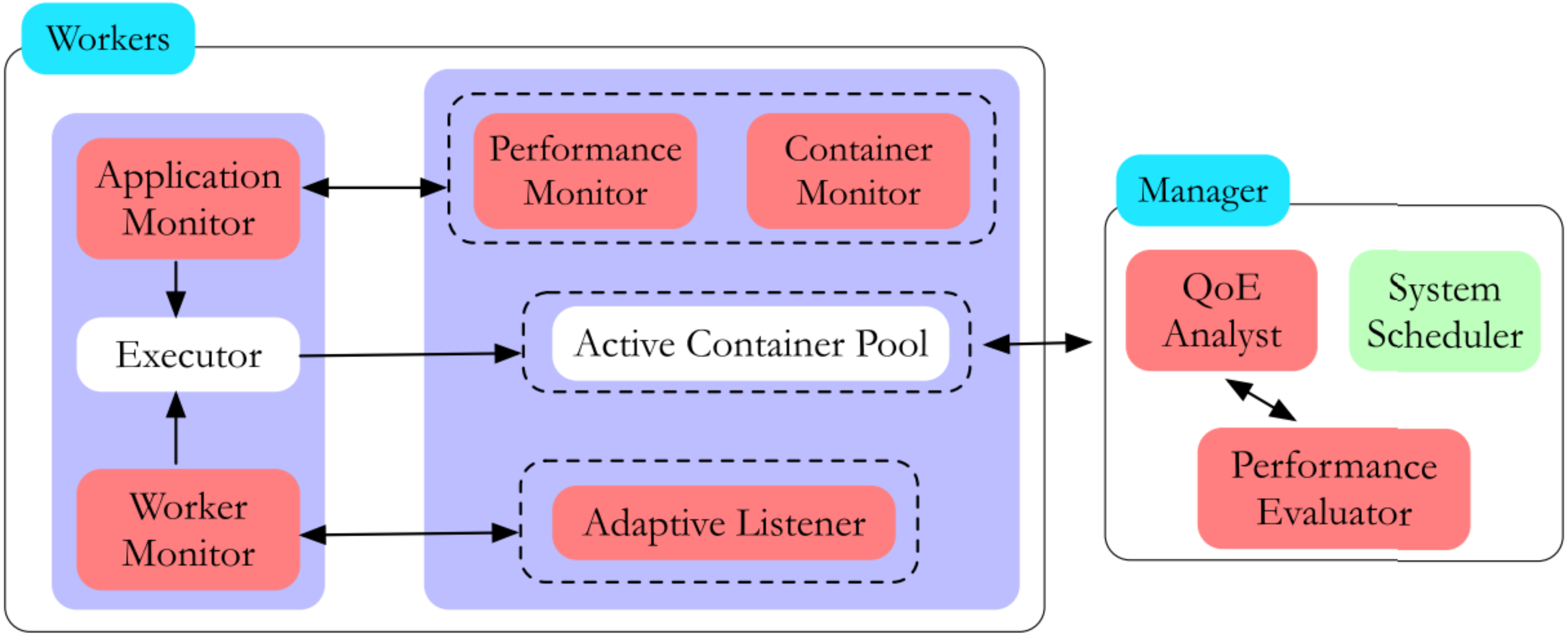
PROPOSED SYSTEM – DQOES

- **DQoES (Differentiated QoE Scheduler) addresses this gap:**
  - Users specify QoE targets (like response time).
  - DQoES dynamically assigns or limits resources accordingly.
- **All models run in Docker containers on shared cloud infrastructure.**
- **Designed to:**
  - Be non-intrusive (no changes to underlying cloud platforms).
  - Work with heterogeneous workloads.

SYSTEM ARCHITECTURE

- **Based on Docker Swarm (manager-worker model):**
  - **Manager Node**:
    - Collects QoE targets.
    - Monitors overall system performance.
    - Makes scheduling decisions.
  - **Worker Nodes**:
    - Host the containers.
    - Track container performance.
    - Apply resource changes.
- **Four Key Modules**:
  - **QoE Analyst** – Receives and analyzes QoE targets.
  - **Application Monitor** – Observes model response times.
  - **Worker Monitor** – Tracks resource usage across containers.
  - **Executor** – Applies CPU/memory adjustments per container.

SYSTEM ARCHITECTURE



DQoES System Architecture

PERFORMANCE MANAGEMENT ALGORITHM

- **DQoES classifies containers into three sets:**
  - **G (Good)** – Performing better than target.
  - **S (Satisfied)** – Meeting the target.
  - **B (Bad)** – Performing worse than target.

- **Adjusts resources based on classification:**
  - Reduce resources for G to free up capacity.
  - Increase resources for B to help them reach the target.

- **Optimization Goal:**
  - Maximize the number of containers in **S**.
  - Ensure system-wide **QoE balance** under resource constraints.

PERFORMANCE MANAGEMENT ALGORITHM

Initialization: W_i, $c_i \in C$, $o_i \in O$, $p_i \in P$,
for $c_i \in W_i$ **do**
    $R(c_i, t) = r_i$
    $P(c_i, t) = p_i$
    $q_i = o_i - p_i$
    if $q_i > \alpha \times o_i$ **then**
        $G.\text{insert}(c_i)$
        $Q_G = q_i + Q_G$
        $R_G = r_i + R_G$
    else if $q_i < -\alpha \times o_i$ **then**
        $B.\text{insert}(c_i)$
        $Q_B = q_i + Q_B$
        $R_B = r_i + R_B$
    else
        $S.\text{insert}(c_i)$
for $c_i \in W_i$ **do**
    if $c_i \in G$ **then**
        $L(c_i, t+1) = L(c_i, t) * (1 - \frac{q_i}{Q_G} \times R_G \times \beta)$
        if $L(c_i, t+1) < \frac{1}{2 \times |C|}$ **then**
            $L(c_i, t+1) = \frac{1}{2 \times |C|}$
    else if $c_i \in B$ **then**
        $L(c_i, t+1) = L(c_i, t) * (1 + \frac{q_i}{Q_B} \times R_G \times \beta)$
        if $L(c_i, t+1) > T_R$ **then**
            $L(c_i, t+1) = T_R$

ADAPTIVE LISTENER

- **Collecting performance data continuously is expensive.**
- **DQoES uses an adaptive listener:**
  - Reduces monitoring frequency when system is stable.
  - Reacts quickly when:
    - New jobs arrive or
    - Performance degrades.
- **Uses exponential backoff to control update frequency.**
- **Ensures low overhead and high responsiveness.**

SYSTEM IMPLEMENTATION

- **Implemented as a plugin for Docker Swarm.**
- **Uses Docker CLI tools (e.g., docker stats, docker update) to:**
  - Monitor usage.
  - Apply soft CPU/memory limits.
- **All operations are done without modifying the core Docker engine.**
- **Supports both TensorFlow and PyTorch inference jobs.**

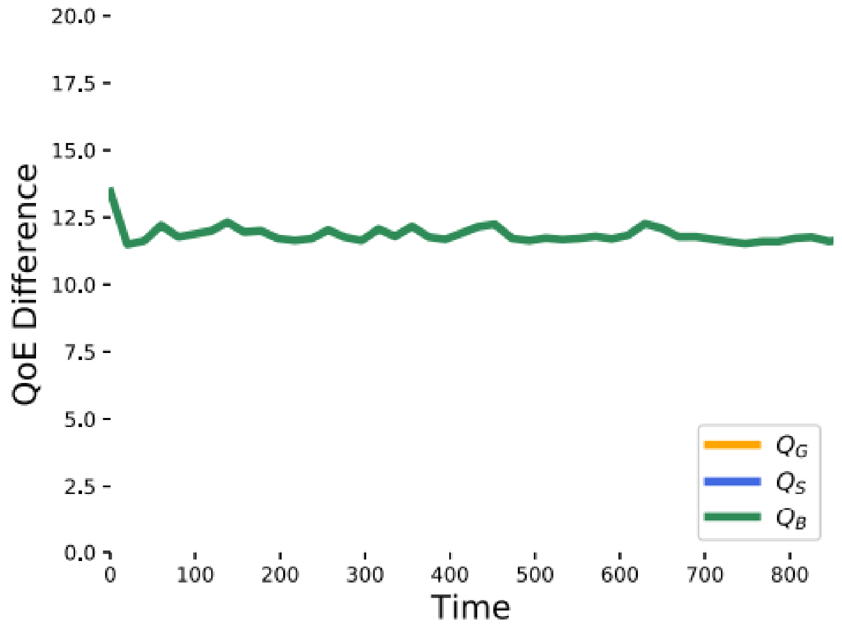
EXPERIMENTAL SETUP

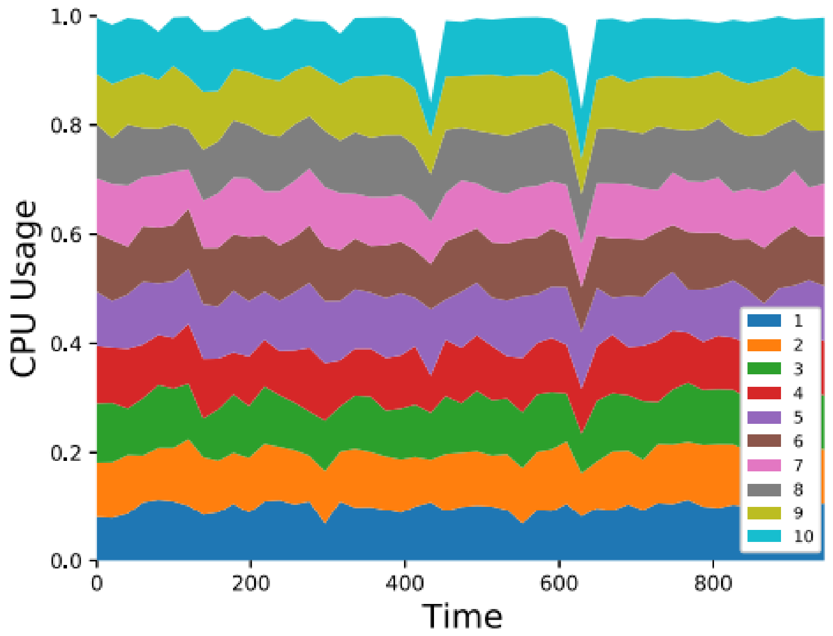
- **Deployed on NSF CloudLab:**
  - Machines with Intel Xeon CPUs and 64 GB RAM.
- **Models Used:**
  - VGG-16, ResNet-50, NASNet, Inception V3, Xception.
- **All inference jobs run in containers.**
- **Each job is a batch of 100 image inferences.**
- **QoE Metric: Response time per batch.**
- **Compared DQoES with default Docker Swarm scheduler.**

RESULTS – SINGLE MODEL (FIXED OBJECTIVES)

- **Scenario 1: All containers have unachievable targets → all fall in class B.**
- **Scenario 2: All containers have achievable targets:**
  - DQoES gradually adjusts CPU shares.
  - Eventually, all containers move to class S.
- **Shows DQoES's ability to reach stable, QoE-compliant state.**

RESULTS – SINGLE MODEL (FIXED OBJECTIVES)



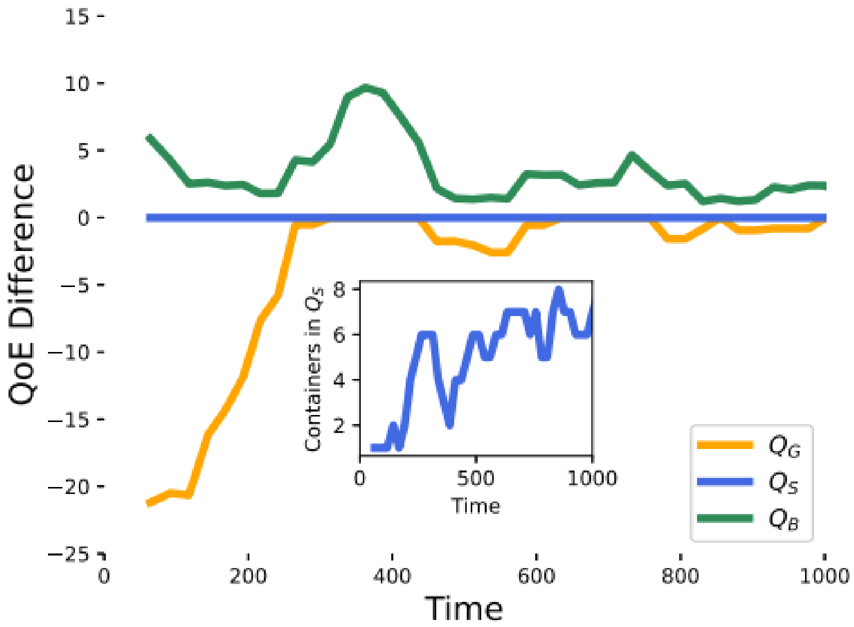


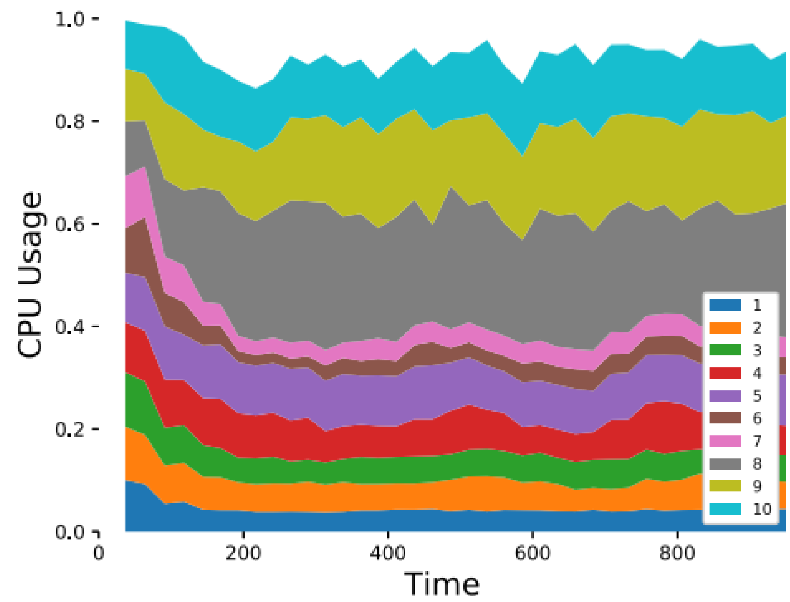
Delivered QoE & CPU Distribution (Unachievable Objectives)

RESULTS – VARIED OBJECTIVES

- **Each container has a unique QoE target (some achievable, some not).**
- **DQoES:**
  - Prioritizes underperformers intelligently.
  - Ensures max containers in class S.
- **Result:**
  - Much better overall QoE.
  - Fairer and more balanced resource usage than default system.

RESULTS – VARIED OBJECTIVES





Delivered QoE & CPU Distribution (Varied Objectives)

RESULTS – CLUSTER ENVIRONMENT

- **40 containers across 4 worker nodes.**

- **Compared DQoES vs Docker Swarm default:**
  - DQoES: 26 containers in class S.
  - Default: Only 3 containers in class S.

- **DQoES adapts to:**
  - Varying workloads.
  - Dynamic job arrivals.
  - Diverse QoE requirements.

CONCLUSION

- DQoES enables cloud clients to define QoE targets for their applications.
- Works on shared infrastructure (multi-tenant).
- Adjusts resources at runtime to meet goals.
- Outperforms Docker's native scheduler in:
  - User satisfaction.
  - Resource efficiency.
- Achieves up to 8x better QoE compliance.

FUTURE WORK

- Introduce fairness mechanisms across users/containers.
- Add container migration based on real-time worker load.
- Extend to other resources like memory and network.
- Explore integration with Kubernetes for broader applicability.

REFERENCES

- Mao, Y. et al. “Differentiated QoE Scheduling for Deep Learning Inferences with Docker in the Cloud.” IEEE TCC, 2023.
- Docker: https://www.docker.com
- TensorFlow Models: github.com/tensorflow/models
- PyTorch Models: pytorch.org/docs
- CloudLab: cloudlab.us
- Google AdSense: adsense.google.com
- Apple FaceID: support.apple.com

THANK YOU