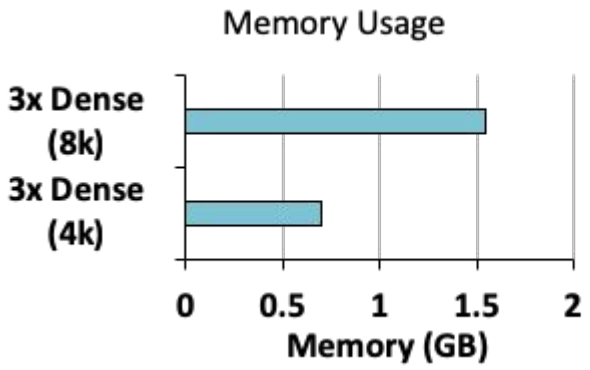
Adaptive Deep Learning on IoT for better Data Integrity

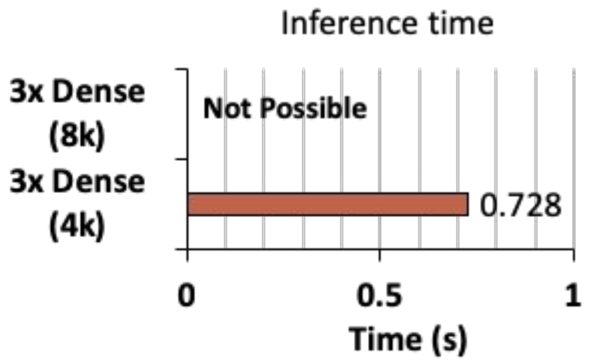
CS6235 Real-Time System Project
Jiashen Cao

Overview

Background

- Deep learning is popular nowadays and is used everywhere
- Deep learning model is heavy
  - Model has large memory footprint
  - Each layer in the model has heavy computation





Background (cont'd)

- Proposed a distributed way to run heavy deep learning neural network on multiple IoT devices

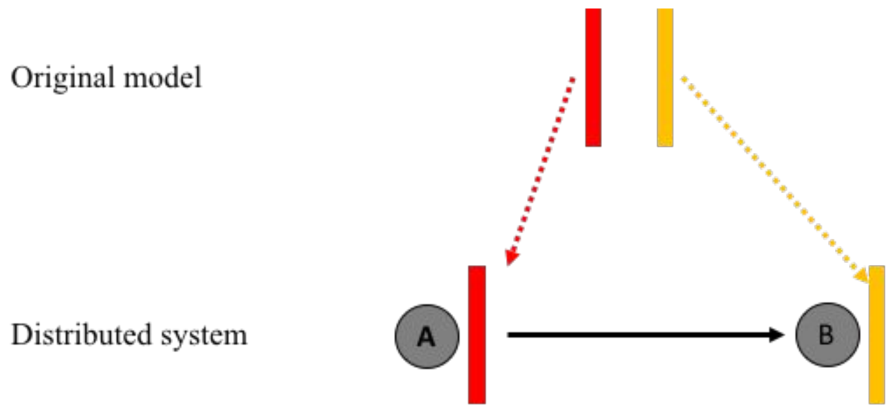


Figure 1

Let's assume we have 2 IoT devices available and a model with 2 layers. We can distribute one layer to one device and form a pipeline system with RPC communication.

Motivation

- The distributed system faces issue of dropping data because data processing rate on each device is different

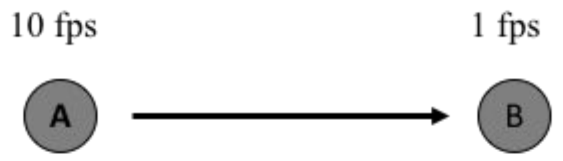


Figure 2

Device A has faster data processing rate than device B. In real time, because device B is only able to process 1 data packet per second, it will drop 9 out of 10 packets from device A.

Introduction

- Propose a adaptive way for running deep learning model
  - In Real Time
  - Focus on prediction instead of training
- Utilize shared resource in the system to achieve
  - Better data integrity coverage
  - Similar or even better performance compared to baseline

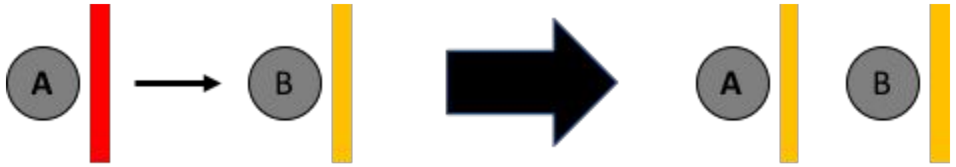


Figure 3

Overall Design

- A server implemented in Python to handle logic of switching task and monitoring overload
- A RPC message protocol for device to device communication
    - Actual deep learning data
    - Unique data identifier for evaluation
    - Overloading feedback
- Deep learning model implementation in Keras

RPC Message Design

RPC communication between different devices

Boolean message(Bytes input, Int identifier);

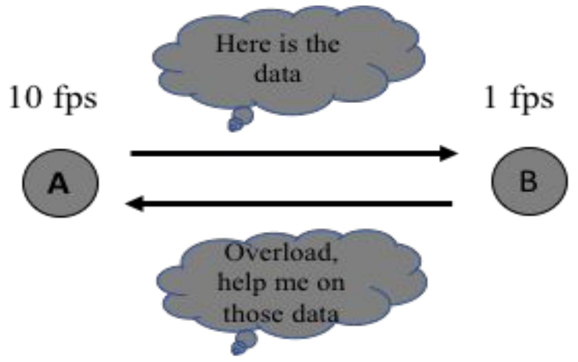
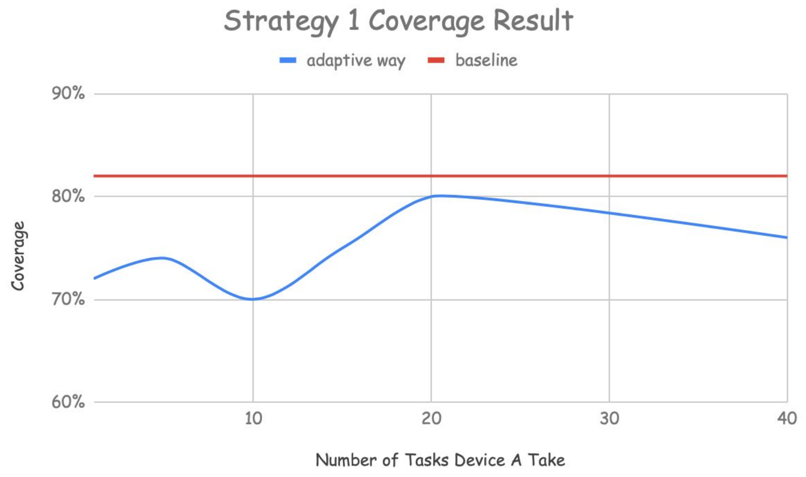


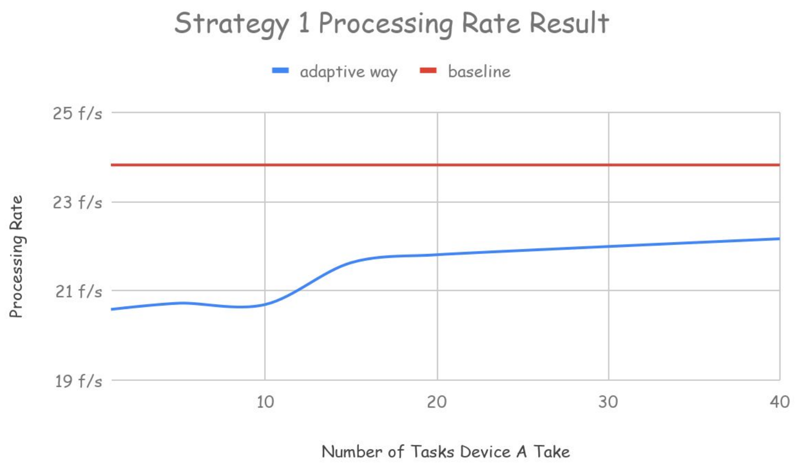
Figure 5

Server Logic Design

- Propose 3 strategies to help device B to leverage the overloading issue
- Strategy 1:
    - After getting overloading message from device B, device A switch to device B’s task
    - Stop sending data packet to device B to reduce its pressure
    - After performing X number of tasks, device A switches back to its own model and resume sending data to device B

Strategy 1 Evaluation



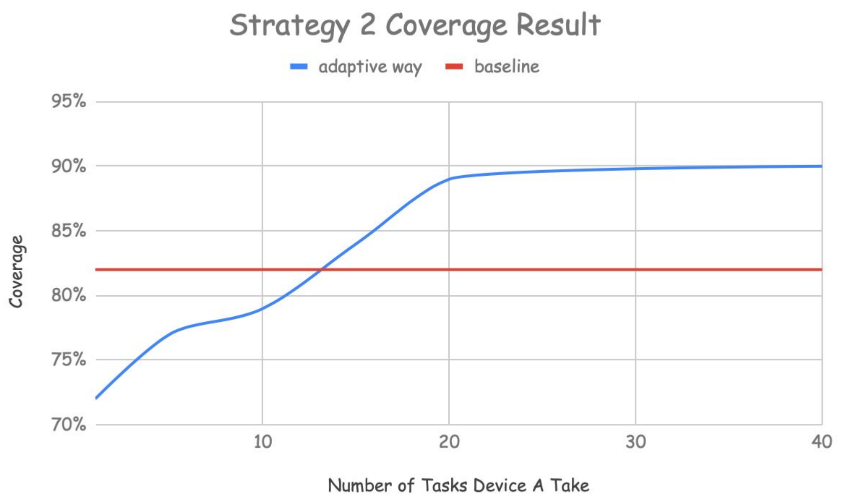


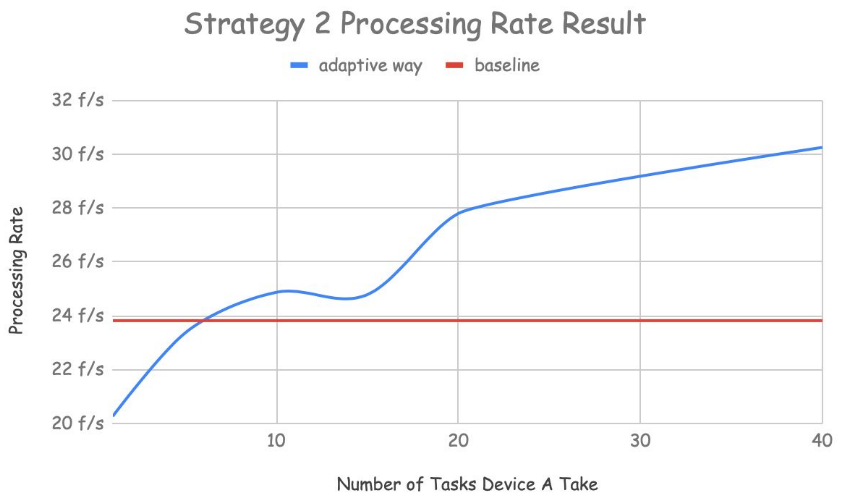
- The proposed strategy has worse coverage and processing speed compared to the baseline

Server Logic Design (cont’d)

- Strategy 2:
    - After getting overloading message from device B, device A switch to device B’s task
    - Sending data packets to device B in reduced speed to reduce its pressure (device A perform 1 computation and then send 1 data packet to device B)
    - After performing X number of tasks, device A switches back to its own model and resume sending data to device B in normal speed

Strategy 2 Evaluation





- The strategy 2 has worse stats if number of tasks taken is small (task switching overhead)
- The proposed strategy has better coverage and also improved processing speed compared to baseline

Server Logic Design (cont’d)

- Naive Strategy 3:
  - Simply group all layers together and place copy on both device A and device B
- Result
  - Achieve 100 % data coverage and 30 f / s image processing rate
- Disadvantage
  - This strategy simply utilizes the rich resource but it does not have function to balance load between devices in real time
  - In practical case, a group of layers may not fit into device memory size, which will cause lots of slow overhead

Live Demo

Conclusion

- I successfully implemented RPC message protocol for data communication
- The strategy I proposed gives better data integrity protection and relatively better performance compared to the baseline