CS 6235 - Real Time System

Final Report

Jiashen Cao

Background

With the evolution of the deep learning neural networks, they are able to do many tasks nowadays. Because of the nature of deep neural network, which has large memory footprint and heavy computation, it usually is deployed on computation resource rich devices such as server with graphic processing unit. However, as an individual or artificial intelligence product customers, they may not have access to those large servers. Instead, in the home setting, they have access to low performant devices such as smart home device, mobile phone, etc. Therefore, [1] one solution is proposed that we can utilize a group of resource constrained devices for running deep neural network in a distributed fashion.

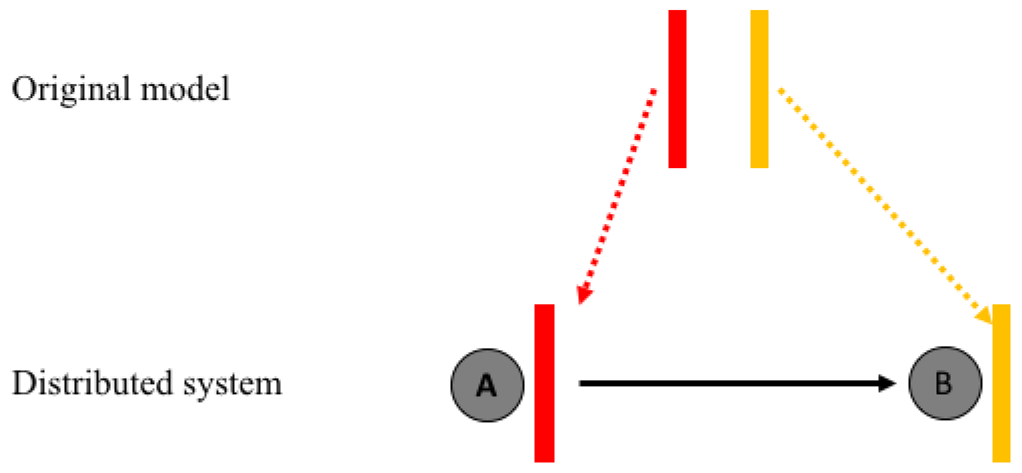


Figure 1

The proposed solution shown in **Figure 1** states that if the system has two layer neural network and two devices available. Each device only needs to deploy one layer and once one device finishes, it then communicates with another device to proceed next layer computation. By doing this way, both devices are able to reduce their own memory footprint and computation pressure. Nevertheless, this approach also presents a drawback that data could lose when it goes through the system. For example, if device B is computing heavy layer and has slower processing rate than device A, it potentially drops data from device A once the arriving queue on device B is full. This issue could hurt the system performance in terms of that system loses important data during communication. In addition, because the device B is slowest node in the system, it becomes the bottleneck which affects the processing rate of the whole system.

Introduction

The motivation of this project is in real world, customers always expect the product to have reasonable results and the speed of the product to be as fast as possible. The original approach proposed is potential to affect the data integrity and also temper the system performance. Hence, my goal of this project is to propose solutions to improve the data integrity and system processing speed and the result is expected to show improvement compared to the baseline approach.

For this project, I want to focus on a small problem which is to reduce the data loss during the system communication and also achieve relatively better data processing rate compared to the baseline. Because device B is much slower compared to device A, it drops certain amount of data from device A. In other word, device A actually wastes its computation resource because some portion of its data is not used by the device B. My idea is that I can implement a feedback message protocol for device B to inform device A. Once device A notices that device B is overloaded, it starts to help on device B's task instead of keeping sending more data to device B.

The main contributions of this project are I build a system from scratch for running simulation for different approaches and I propose three approaches to improve the system. Two approaches among those proposed solutions of utilizing device A's computation resource to help device B do achieve better data integrity and performance compared to the baseline (the original approach). The detailed implementation and results will be discussed in the later sections.

Implementation and Design

Architecture setup

Because my goal of this project is to run deep learning models on resource constrained devices, I choose to use Raspberry Pi 3 as my setup. It has one gigabytes memory size and 1.4 GHz quad-core, which has similar storage space and computation resource as other IoT devices. I believe this architecture will give similar performance output as other IoT devices. All the Raspberry Pi used are configured in Ubuntu Linux distribution because it has well support for Python package managements.

Communication between devices

All devices are located in a local area network and all of them are connected by ethernet to reduce the impact from communication overhead. In order for different nodes to communicate with each other about overloading information, a remote procedure call protocol is used for communication. In the message header, it contains data, an unique identifier for each packet and a flag as return result. One device could easily turn on the flag to indicate other devices that it is overloaded. The unique identifier in the message header is used for evaluation stage that I can calculate how many packets are lost during the transmission.

The remote procedure call is implemented on HTTP message transmission, so I build a multithreaded HTTP server in Python to listen traffic on port 12345. When the system starts, the server will start running and initialize the assigned deep learning neural network. Once the server receives a request, it will put the request onto incoming buffer and the deep learning model constantly extracts data from the buffer and does the machine learning computation. If the incoming buffer is full, the server then drops data. Once the server finishes one data computation, it will communicate with next device in the system.

Deep learning models

For the simplicity of the simulation, I choose to run experiments on two simple linear layers from state of art deep learning model to test my proposed solutions. Both layers have parameter size of 2000 but layer 1 takes input size of 100 and layer 2 takes input size of 2000. I tested the computation time for both layers on Raspberry Pi. The first one takes 0.004 seconds and the second takes about 0.04 seconds, so the second is 10X slower than the first one. I use the second layer for simulation the slower task and the first layer to simulate on faster task. The system I simulates focuses on prediction instead of training.

System setup

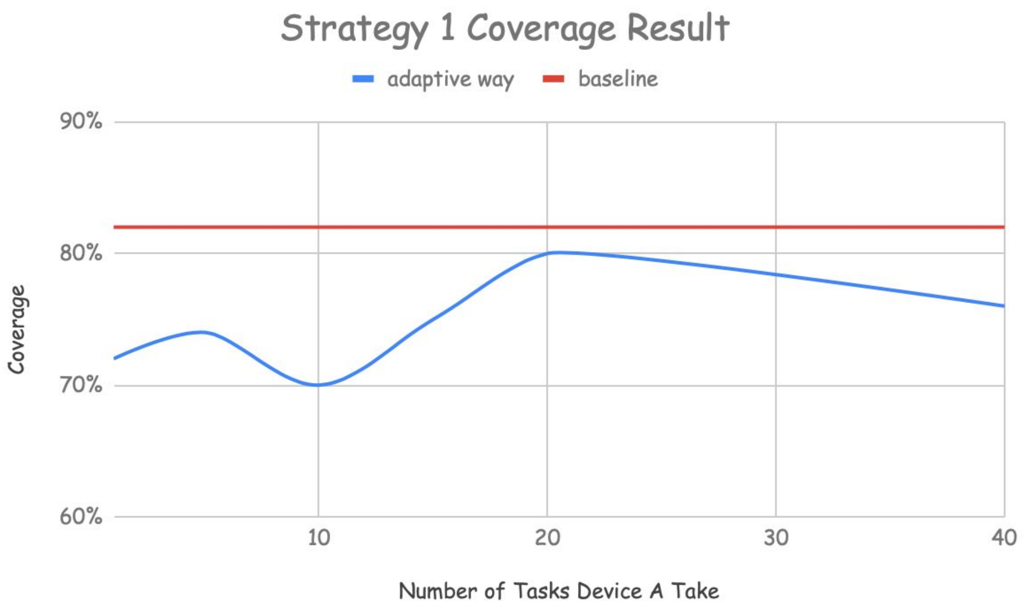
The system typically contains 4 devices, 1 client node, 1 fast node, 1 slow node and 1 end node. The faster layer from previous discussed section is deployed on fast node and the slower layer is deployed on slow node. The pipeline works as the client node sends data to fast node, fast node sends data to slow node, slow node sends data to end node and end node prints out all statistics including data coverage and data processing rate. During the run time, the slow node is able to detect whether it is overloaded or not and then it sends feedback to the fast node. The fast node will make decision to help the slow node accordingly.

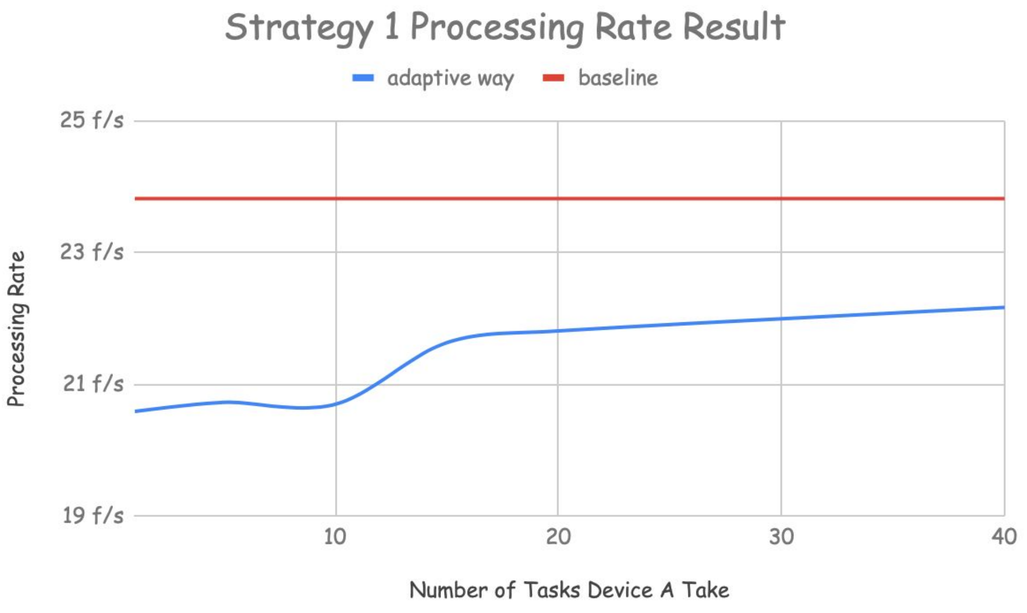
Server Logic Design

I will discuss my three proposed ways to reduce workload pressure on slow node in this section and I will also share performance results to compare with baseline approach.

Strategy 1

The idea of strategy 1 is once the slow node is overloaded, it will send feedback message to the fast node. Then the fast node switches to slow node's model and starts to perform computation for the slow node. Ideally, this approach should give better data coverage and data processing rate because data is not sent to slow node anymore when it is overloaded. In other word, it could reduce the data dropping at slow node. However, I simply ignore another factor that in the real system setup, once the fast node starts to do computation for slower layer, it becomes the new bottleneck for the system. The fast node itself starts to drop data. In addition, because the server needs to switch model during run time, it introduces extra context switching overhead.

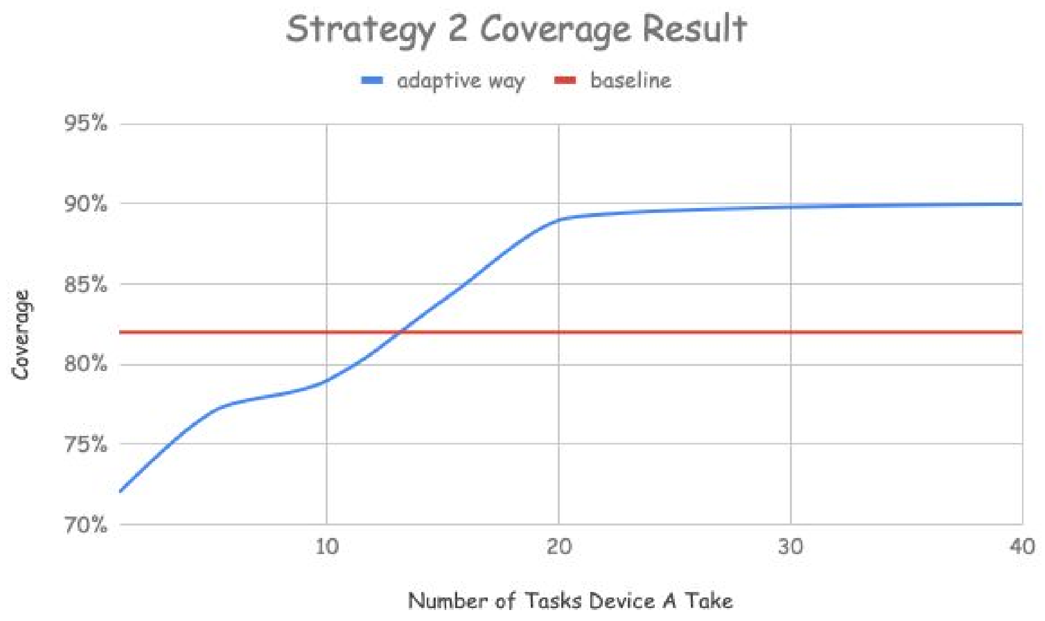


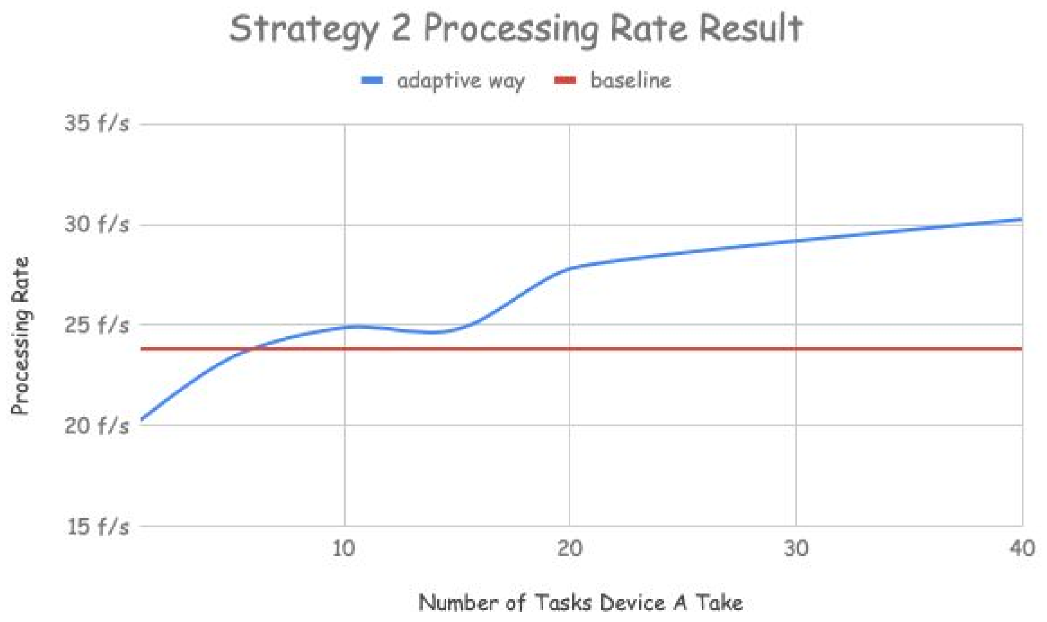


Those two figures above are the strategy 1 evaluation result. Because of the issue I mention, we can conclude from the results that strategy 1 actually performs worse than the baseline.

Strategy 2

Based on the disadvantage of strategy 1, I rethink my approach and propose another strategy for the system. The difference between strategy 2 and strategy 1 is that the fast node in strategy 2 still sends data to the slow node if slow node is overloaded, but it sends in a reduced speed. The key idea is the fast node will send one data to the slow node after it finishes one computation. The reason I design this logic is because fast node and slow node are doing the same computation, which means they have same processing speed. When the fast node finishes its computation, the slow node is also about to finish one computation, which means the buffer on slow node will potentially have one slow available. Therefore, sending one data from fast node to slow node at that time has lower possibility of being dropped. I also evaluate this strategy in terms of data coverage and data processing rate.





The results show that the strategy 2 has better data coverage and data processing rate once the number of tasks taken by fast node is beyond some thresholds. Number of tasks taken by device A (fast node) means number of tasks performed after fast node switches to slow node's model. We can imagine that if that number is small, fast node spends lots of time on overhead for model switching, so in that case there is no benefit of using strategy 2. As number of tasks taken increases, the performance starts to improve because the fast node starts to perform more useful work. In addition, we can tell from the graph that the performance stops to increase at a certain point. I believe the reason is because the overhead of model switching is huge, data loss during model switching period cannot be avoided.

Strategy 3

For strategy 3, I simply aggregate all layers into one single model and copy the model onto both fast node and slow node. The client in this strategy needs to communicate with both fast node and slow node and they communicate with only end node. This strategy gives 100 % data coverage and 30 f / s data processing rate.

However, I think there are a few drawbacks associated with this approach. First of all, it requires fine tuning the system during downtime especially when the deep learning neural network becomes complicated. Because I simply pick 2 linear layers from deep neural network to simulate, strategy 3 works well only for this case. Because this strategy does not distribute the model, it is hard to make change adaptive during real time. In addition, once the parameter size of a layer increases, the aggregation of multiple layers may not fit into the device's memory, which causes lots of hard disk overhead.

Experiments

All experiments are included in my code, and I provide a detailed README file for setting up the environment and running simulation experiments.

Future Work

For this project, I use a group of Raspberry Pi in a local area network connected by ethernet, in which device has less communication overhead and energy consumption compared to other network protocols. However, because my target for this project is to use a group of IoT devices and those IoT devices are typically connected by WiFi, in the future, it is useful to analyze the system in a WiFi network setting. In addition, energy consumption is another big concern for edge devices and WiFi network usually consumes more energy than ethernet. Hence, it is also interesting to analyze the system from the energy consumption perspective.

Another thing could be improved is that currently I am using a binary feedback message, which tells client whether it is overloaded or not. The protocol succeeds in this system because I am using the exactly same devices across the system. However, in order to make the system more robust that it could be adapted to all different devices, I can improve on the feedback message protocol to include more meaningful data. For example, it can provide the computation time on different devices.

Conclusion

To conclude this project, I am able to build a system scratch by using external RPC protocol, utilizing Python HTTP server and coding the sample deep learning model. My main contribution is 3 proposed solutions for running the whole deep learning distributed system in a adaptive way. The first strategy does not present any benefit compared to the baseline, but it does serve a foundation for me to think about other 2 strategies. And those 2 strategies give better data coverage and data processing speed compared to the baseline.

References

[1]: R. Hadidi, J. Cao, M. Woodward, M. Ryoo, and H. Kim, "Musical Chair: Efficient Real-Time Recognition Using Collaborative IoT Devices," ArXiv e-prints:1802.02138.