**An Adaptive Collaborative Computing System for Distributed Deep Learning**

***Abstract*—**

***Index Terms*—OpenCL, DNN, embedded device**

**## Introduction**

Based on the Distributed Deep Neural Network (DDNN) described by H.T.Kung et al., we propose Distributed Deep Neural Network Computing Framework(DDNNCF) an adaptive collaborative computing system that can be used for deep learning. DDNNCF distributed a single DNN onto hierarchies composed of end devices and servers, with the support of OpenCL, the model could run on parallel hardware. The framework designed on dataflow programming paradigm to offload computation partially to server and increase system efficiency. All the devices connected to a server will share the resources of the server to complete a single DNN, how to allocate the resource assignment, reduce the work latency or increase the system throughput are very important. We use hardware supported virtual GPU (vGPU), such as NVIDIA GRID and AMD FirePro S7100 GPUs, to make resource independent avoid the interference of different work. DDNNCF also has modular response for dynamically assigning resource to different work to improve system efficiency. By leveraging the threshold got from training, the conflict classification can exit the end device, to some extent, reduce network transaction and server power consumption.

In a nut shell, DDNNCF has the following features:

i. DDNNCF adds OpenCL support, enable eBNN-based inference for diversified HW platforms.

ii. DDNNCF designed on dataflow programing paradigm. In virtual of Chainer’s feature, DDNNCL can realize hierarchical inference computation, what’s more user can develop specified deep learning model on Chainer.

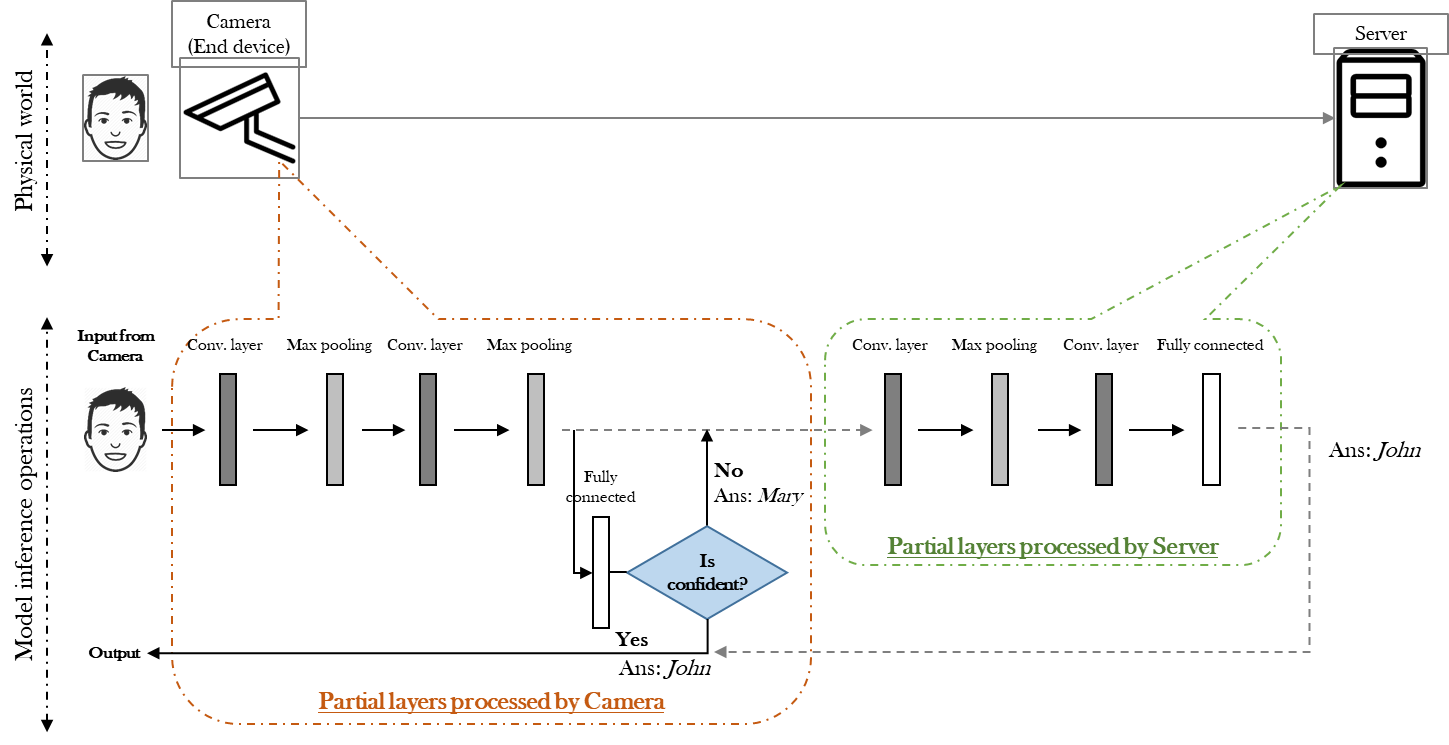
iii. By using the vGPU, DDNNCF can dynamically assign the resource of server to the connected end devices.

iv. For specific IOT application, the end devices and servers only do the inference. According to the threshold generate in the training, the inference can be fast and accurate.

**隨著深度學習的不斷發展，及其具有了人工智慧（AI），同時IOT的出現使得機器之間可以互相連接。當透過IOT相連的device之間具有AI的時候，有可能將人類從繁重的工作中解放出來。**

**儘管有以上的優點，但是將DNN帶入到IOT中會增加終端裝置的計算負擔。敘述DNN并點明終端系統的storage是其瓶頸。**

**為了解決該問題有人提出eBNN.介紹eBNN之後，說明儘管eBNN使得small end device可以運行DNN且在手寫辨識中取得不俗的成績，但是並不能很好的適用於real application。[XXX: We did not focus on the model design.] 為此我們基於DDNN分佈式計算的思想[XXX: You mentioned that our work is based on DDNN, but DDNN is not introduced properly.]，提出DNNCL an adaptive collaborative computing system that can be used for deep learning. 介紹DDNNCL的功能和技術。**

****

**Fg。1**

**總結DDNNCL的feature（2）design on dataflow programming paradigm [XXX: ???]（1）加入opencl支持[XXX: we allow the use of accelerator for boosting inference performance.] (3）使用vGPU完成dynamic job assign（4）基於MQTT協議傳輸數據 [XXX: provide high-level description.]**

**In the rest of the paper, Section 2 motivates our DDNNCF[XXX: if we add motivation example Fg 1, we might need to do experiments.], Section 3 introduces the architecture and key components of DDNNCL framework. Section 4 is some experiments to verify the proposed DDNNCL, followed by results evaluation in Section 5. Section 6 concludes this work and future work.**

**##2. Background and Motivating Example**

DDNN, entropy and early exits.

**##3. Framework**

In this section, we will give an overview of the proposed distributed deep neural network computing framework (DDNNCF) and describe how it works.

**\*\*A. DDNNCF architecture \*\***

The increased performance makes DNN model becoming much more complex, as a result, causes a lot of additional latency and energy costs. DDNNCF supports adaptive inference computing that end devices can dynamically compare the prediction results with the pre-trained threshold, the high accuracy result would exit locally otherwise would send to server for help. Multiple devices.

For example, a face recognition system is a computing application capable of identifying a person from an image or a video record by a camera device. There are four stages in the lifecycle of a face recognition system based on DNN, as shown in Figure 1. Firstly, pre-training the model and sending it to end devices and server. Secondly, Checking the inference result in the end device and comparing with a preset threshold. Exiting the prediction whose entropy above threshold means high accuracy, would reduce network communication and power consumption, improve the performance as well. Thirdly, the unconfident prediction will send intermediate result and layer number to the server to continue computing. Fourthly, the final classification computed by the server will send back to end device.

The architecture of DDNNCF is illustrated in Figure 2 consist of two parts. one is Device module running on end devices to enable the pre-trained DNN model execution and provides communication service, and the other one is Server Module running on backend server which is responsible for remote model execution and communication. When there are multiple different end devices send computing requests to the server at same time, Server Module would assign resources to these tasks and compute them parallelly.

In the following subsection, we describe the detail of key features presented in DDNNCF.

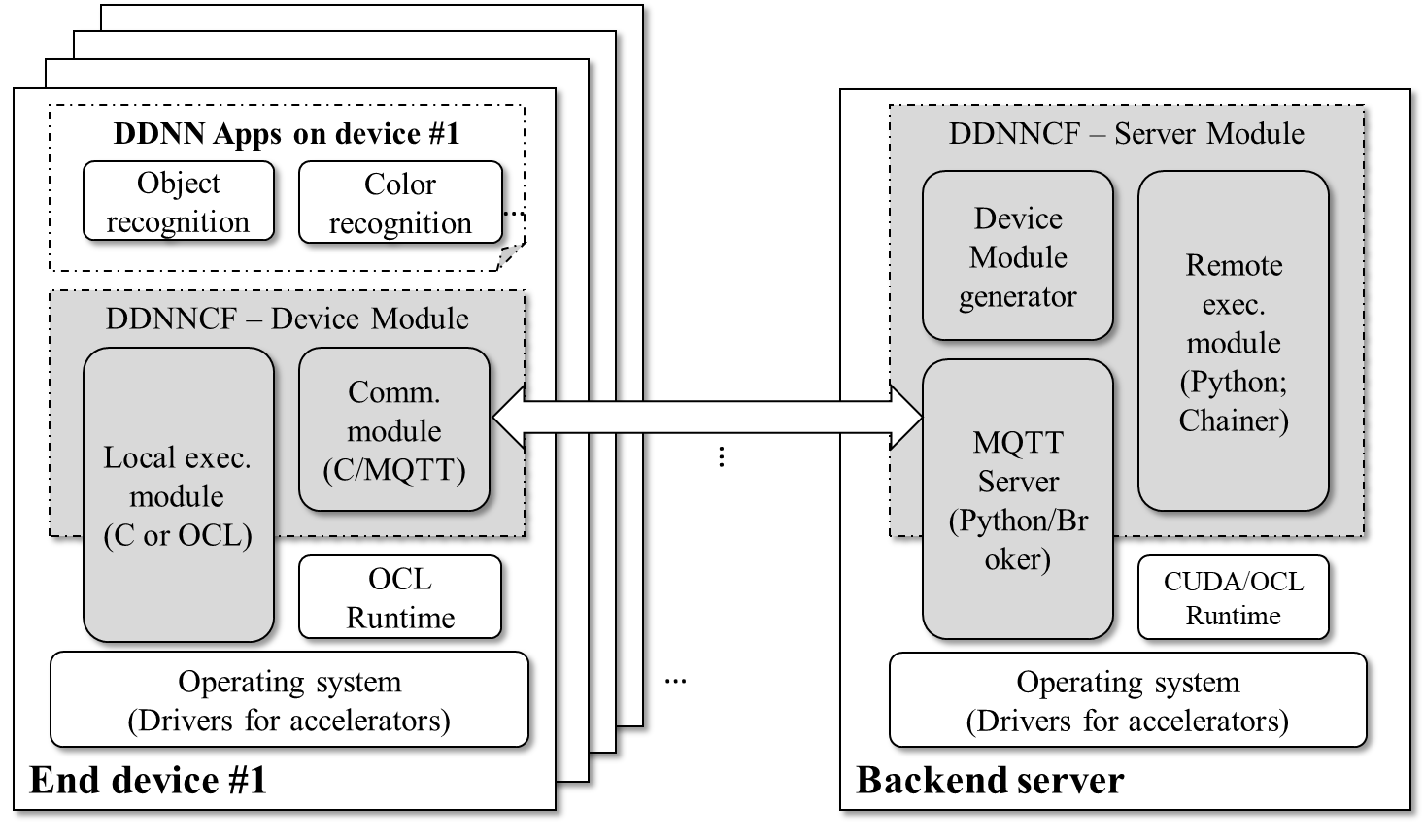
****

Figure 2. DDNNCF architecture

**\*\*B. DDNNCF – Server Module \*\***

DDNNCF Server Module contains three parts, Device Module generator, Remote execution module, and MQTT Server, responsible for C/OpenCL model generate, cloud computing, and network communication, respectively

* Device Module generator would generate the trained model in C/Python/OpenCL code to accelerate the inference and support different devices, then send to Remote execution module and Local execution module separately.
* The Remote execution module is based on Chainer, which is a “Define-by-Run” deep-learning framework can build a block of model definitions and link with the list-like interface. The server can according the parameter send from end device to get the intermediate output, used model, and the DNN model, then server will help to do the further computation; this intermediate execution is supported by Chainer.
* The MQTT Server would publish client requests to Remote execution model, after further inference, it would return the predictions to clients as final classification. The details of how server communicate with end devices will be discussed in the section 3-D.

**\*\*C. DDNNCF – Device Module \*\***

Device Module handles the input (e.g., an image or a video captured by the camera on end device) and make a classification. Local execution module get the C/OpenCL model generated by the Device Module generator to do inference and optimize the code using accelerator for boosting inference performance. Local execution model also need to check the intermediate output during the inference to determine whether the result is confident or not, confident one would regard as the final classification and exit locally, or else would use Communication module sends to server for further computation. Since the end device need to know final classification, Communication module also receives the results from server. We designed intermediate output store in binary format to be smaller than the input, therefore the communication cost between end devices and server will be drastically reduced. The details of communicate will be discussed in the section 3-D.

**\*\*D. Communication of DDNNCF Inference\*\***

During the DDNNCF inference, unconfident prediction in the end devices need send to server for further computation. Since the face recognition need to be classified in real time and the memory of embedded device has limitation, we propose the communication between end devices and server follows MQ Telemetry Transport (MQTT) which is an open protocol works as lightweight publish/subscribe messaging transport.

**D-i Why the communication based on MQTT?**

Compare with the existing communication protocols, MQTT is lightweight protocol for connections with remote locations which is suitable for the environment of our applications. With the emergence of topic, MQTT can easily broadcast the message to a bunch of people eliminate the

--------------------------------maybe survey MQTT paper

**D-ii Workflow of Comm. Module and MQTT Server**

There are three stages during the inference between end devices and server: (1) initialization, (2) communication, and (3) finalization. Take one thread inference as an example, the workflow is shown as Figure 3 and described as follows.

1. The communication starts from an end device sends a CONNECT request to server, then after receiving the request server would send a CONNACK back to implementing that the connection has been established. Since Broker need assign computation resource for the computation requests from end device, use Thread-per-Message to create thread and finish the computation would cause a lot of costs, we take the idea of thread pool for achieving concurrency of the execution. Each time a computation request come, Broker will assign a thread for it.
2. After the initialization process, end device can subscribe to the TOPIC2 to get the final classification, and server will return a SUBACK in response. End device would encapsulate TOPIC1 with message (e.g., DNN model, layer number, and intermediate result) and send them to server for further computing. Broker will choose thread in the thread pool, assign computation resource for it. After the thread SUBSCRIBE to TOPIC1, it will receive the message send from end device in TOPIC1. Combining the feature of Chainer, server will accomplish the model inference and send back the result in TOPIC2. End device can SUBSCRIBE to TOPIC2 to acquire the final classification.
3. The communication ends at the end device has got the classification, where end device would send disconnect to server and get reply.

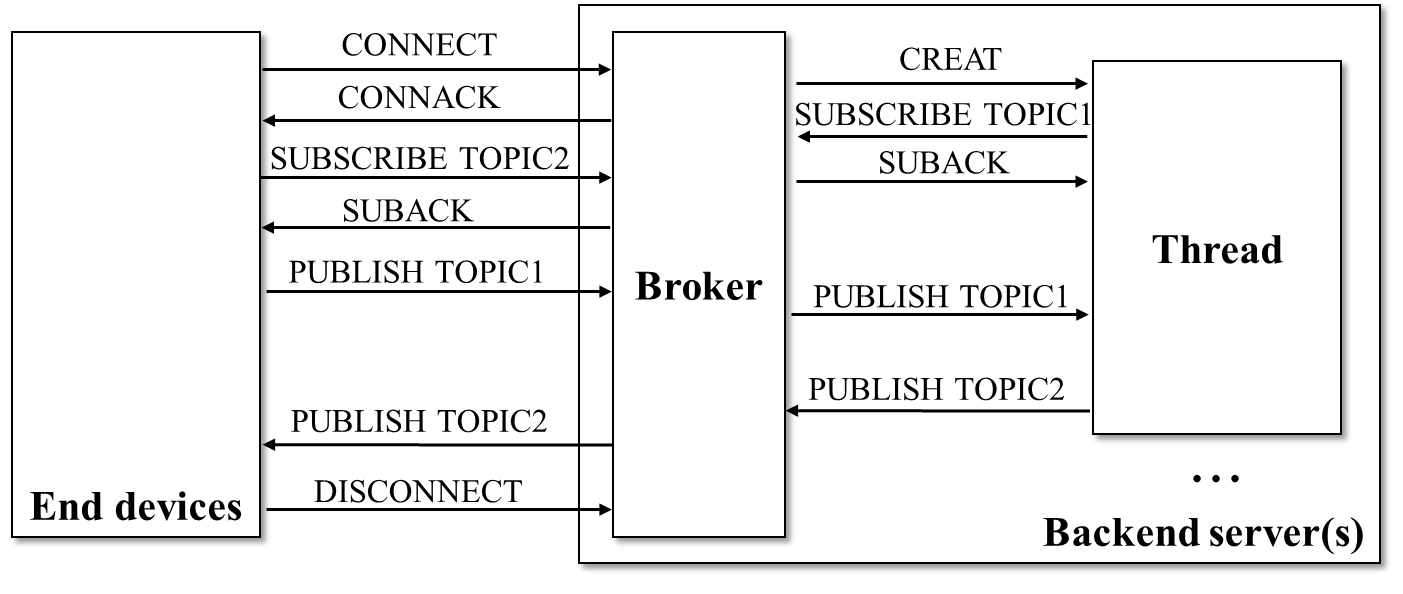
****

Figure 3. Communication between Comm. module and MQTT Server. TOPIC1 represents for the topic of end device requests, and TOPIC2 represent for the topic of the final prediction.

**D-iii error control?????**

SSL/TSL

**D-iv Retained Message**

Retained message is a useful feature of MQTT which can be used for keeping the last state of each topic, when a client subscribes to the topic, broker will send the retained message immediately. Without retained messages, the subscriber would have to wait for the status to change before it received a message. We use this feature to deliver the system configuration, each client that connect with the broker would receive this message and reconfigure the framework. For example, apply this feature to (d) in Figure 3, the retained message will conclude the location of aggregator and system configuration.

**D-v Overheads**

Design experiment to evaluate the time of initial, communication

**Life cycle，issue，work flow, overhead(initialization, publish, subscribe), 网路挂了要怎么办（MQTT feature）error control，thread和end device之间的对应**

**## Experiment and Evaluation**

**## Related work**

**## Conclusion**