Cloud Computing Final Report

Build IoT Platform using Docker, Kubernetes, and Tensorflow

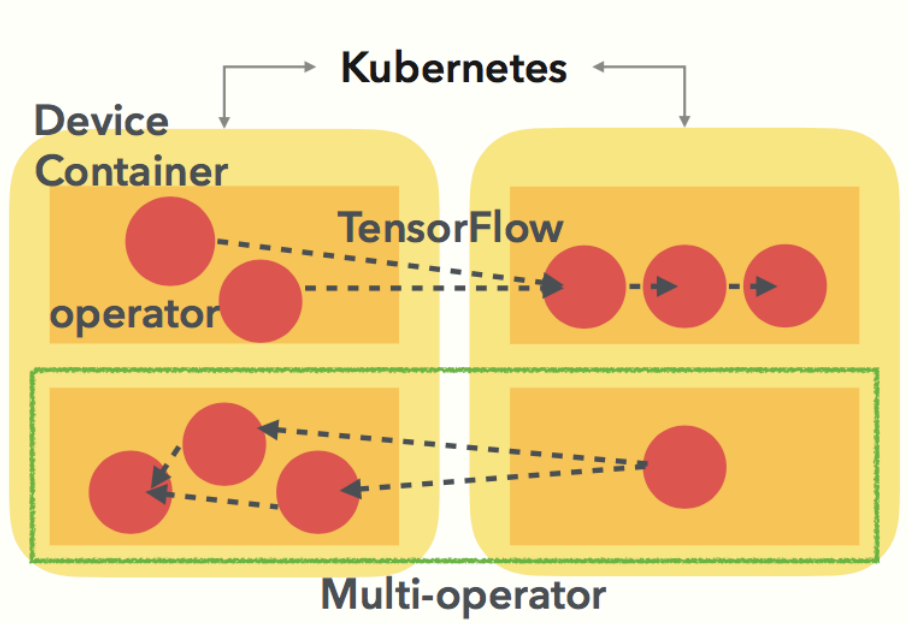
羅允辰

Github Project

我將 final project 的程式碼和實驗結果都整理至 github 上，老師和助教若有興趣可以按造 README.md 來進一步瀏覽，若覺得還不錯也請給我一個星星，謝謝
連結：https://github.com/jasonlo0509/Dockerized-YOLO-on-Rpi-Cluster

IoT Platform Prototype

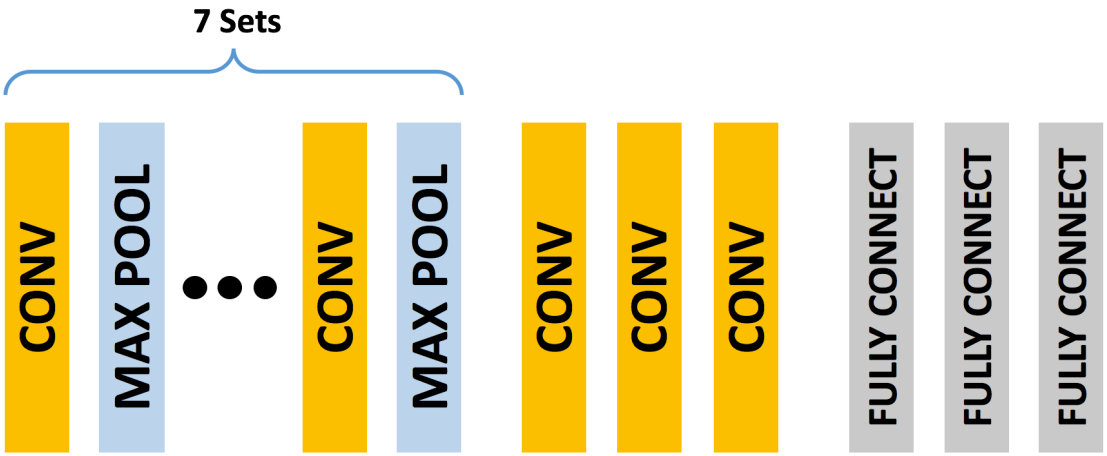
在 "Distributed Analytics in Fog Computing Platforms Using Tensorflow and Kubernetes“ 與 “From Cloud Computing to Fog Computing: Unleash the Power of Edge and End Devices” 兩篇近年的 paper 中，都架設了一個由 3 個 open source projects (Docker, Kubernetes, Tensorflow)所組成的 IoT Platform Prototype。
Platform 的三個 projects 之間關係如圖一所示，一個 application(ex. YOLO object detection) 會先用 Tensorflow 拆成數個 operator，將 operator 與其執行的環境包裝成 Docker Image 後，最後使用 Kubernets 將 Docker Image 部署到指定的 Device 上。



圖一. IoT Platform Prototype

Application : YOLO Object Detection

我們所選的 Application 為 YOLO objection detection ，YOLO 可以判斷圖片中物體的位置與該物體是什麼，適合用在智慧 IoT 的服務上。
我們的簡化版 YOLO 架構用 10 層 Convolution layers 與 3 層 Fully connected layers 所組成，詳細架構如圖二所示。

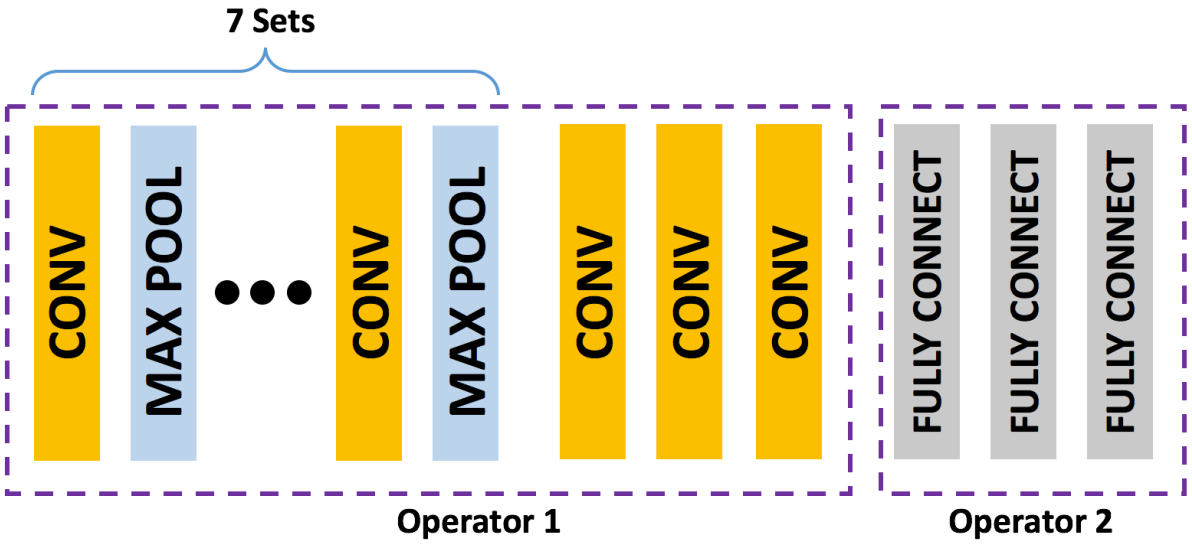


圖二. YOLO Architecture

Application Partition

在拆分 YOLO 時，我做了兩種嘗試，並且在 Mac 上測試 CPU 使用量，分析分別如下：

1) 依據 Layer 特性， Convolution Layers & Fully Connected Layers 各自成為一個 operator，如圖三所示



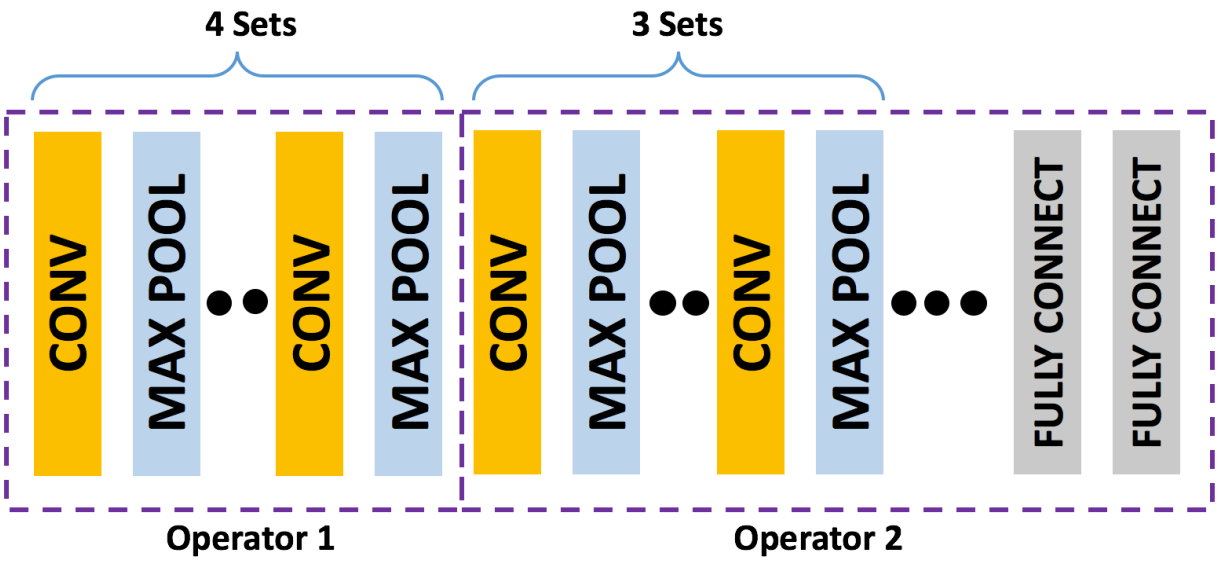
圖三. 1st Partition

但是這種切法會使得兩個 operator 的 CPU 使用量相差太多，如圖四所示

Process Name	% CPU	CPU Time	Threads	Idle Wake Ups	PID	User	
python2.7	0.0	1.12	30	1	3264	jasonlo	
python2.7	4.1	11.74	30	2	3168	jasonlo	Operator 2
python2.7	76.6	13.40	14	4	3888	jasonlo	Operator 1
python2.7	213.5	8:33.98	39	3	3069	jasonlo	

圖四. CPU Resources Usage

2) 按造計算量來平均切分，如圖五所示



圖五. 2nd Partition

我們可以由圖六觀察到，按造這樣拆分，CPU 執行時的兩個 operator 的工作量是差不多的

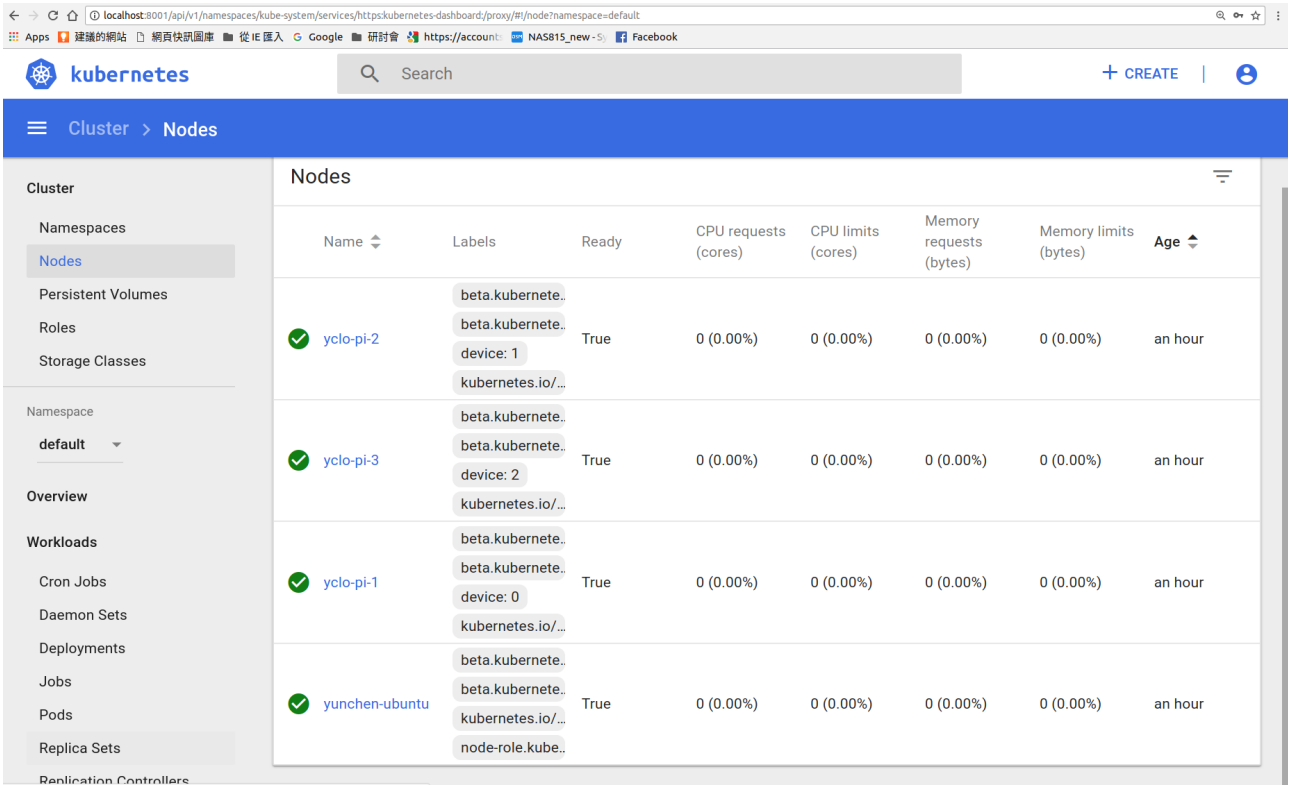
Process Name	% CPU	CPU Time	Threads	Idle Wake Ups	PID	User	
python2.7	14.6	1:00.05	38	1	4482	jasonlo	Operator 1
python2.7	59.1	1:06.10	19	0	4860	jasonlo	Operator 2
python2.7	117.5	5:26.48	31	0	4490	jasonlo	Operator 1
python2.7	125.9	8:43.87	31	1	4485	jasonlo	Operator 2

圖六. CPU Resources Usage

根據以上分析，我們最後選擇的 partition 方式是(2) 按造計算量平均拆分兩個 operator。
如此一來，當 input 不斷進到兩個 operator 中時，才不會因為 workload 不平均而慢下來。

Our Platform

我最終使用了 3 台 Raspberry Pi 作為 IoT Devices，用 wifi 連在一起，並且使用 Kubernetes 將他們連成 cluster，圖七為執行 kubernetes dashboard 時，由 host 端可以觀察到各個 device 情況的圖片。
yclo-pi1~3 為 raspberry pi，而 yunchen-ubuntu 則是 master 負責指派工作給 worker。



圖七. k8s dashboard

Evaluation

由於 wifi 的傳輸速度太慢，無法將 feature map 夠快的傳給下一個 operator，我實驗發現多個 node 的時間會比較慢

因此，我最後的實驗數據是在單個 node 上進行，並且以 CPU 的資源多寡來模擬多個 node 會有的加速。最後我得到在一個 device 上每分鐘可以辨識 19 張圖片，2 個 device 則會加速至 22 張照片。

下圖的橫軸為 device 數量，縱軸為一分鐘可以辨識幾張照片。