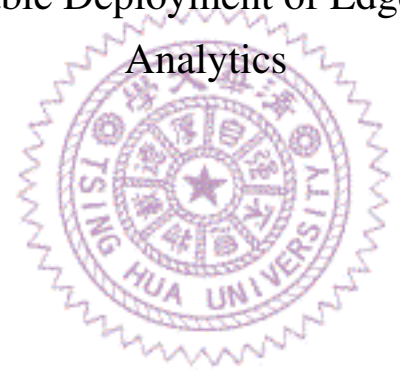
國立清華大學電機資訊學院資訊工程研究所
碩士論文
Department of Computer Science
College of Electrical Engineering and Computer Science
National Tsing Hua University
Master Thesis

動態且可擴展的邊緣物聯網分析應用部署
Dynamic and Scalable Deployment of Edge Internet-of-Things Analytics



蔡霈萱
Pei-Hsuan Tsai

指導教授：徐正炘 博士
Advisor: Cheng-Hsin Hsu, Ph.D.

中華民國 107 年 3 月
March, 2018

國立清華大學
資訊工程研究所

碩士論文

動態且可擴展的邊緣物聯網分析應用部署

蔡霈萱 撰

107
3

國立清華大學碩士學位論文
口試委員會審定書

動態且可擴展的邊緣物聯網分析應用部署
Dynamic and Scalable Deployment of Edge Internet-of-Things Analytics

本論文係蔡霈萱君 (105062508) 在國立清華大學資訊工程研究所完成之碩士學位論文，於民國 107 年 3 月 27 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

_____

_____

_____

_____

_____

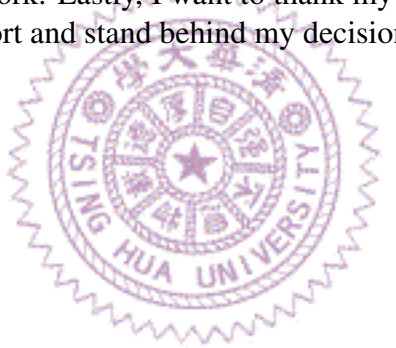
所 主 任 _____

致謝

在此我要感謝在過去兩年中所有幫助過我的人，如果沒有你們的協助我一定沒有辦法順利完成我的論文。在此我要感謝我的指導教授：徐正炘教授，他在我研究的過程當中給予了相當多的指導以及建議，若不是他持續的支持及鼓勵我一定沒有辦法完成如此多的事情以及學到如此多的東西。我也要感謝聯詠科技提供了我一年的獎學金，讓我在研究學習的過程中更無後顧之憂。另外，我也要感謝網路與多媒體系統實驗室的同學們，特別是洪華俊學長，他在過去兩年的研究當中幫助我非常的多。最後，我要特別感謝我的父母，他們提供我堅定不移的倚靠，同時也支持我所作的每一個決定。

Acknowledgments

I would like to express my gratitude toward all the people who helped me in the past two years. I would like to express my sincere gratitude to my advisor Prof. Cheng Hsin Hsu for the continuous support of my research, for his useful comments, remarks and engagement through the learning process of this master thesis. I would like to thank Novatek Microelectronics Corp. for providing me one year scholarship, so I can learn and do my research without worry. Also, I like to thank my labmates in Networking and Multimedia Systems Laboratory, especially Hua-Jun Hong, who helped me a great deal in the course of my research. Without Hua Jun's careful tutoring, I would not be able to finish this work. Lastly, I want to thank my parents for provideing me with the firm support and stand behind my decisions along the way.



中文摘要

近年來，物聯網（Internet-of-Things）應用程式所產生的大量數據皆需要强大的分析方法，例如深度學習，來提取出有用的資訊，而現有的物聯網應用程式大多將數據傳輸到計算資源豐富的數據中心來進行分析。但是，大量的數據可能會造成網絡的壅塞、數據中心的嚴重負擔、以及增加安全上的漏洞。在我的論文中，我設計了一個採用霧計算概念的平台，將數據中心（服務器）和終端運算裝置（物聯網運算裝置）的資源進行整合。它有兩個特點：（i）動態部署和（ii）邊緣分析。我在位於邊端的運算裝置之間執行分佈式的分析應用程式來對數據進行預處理，而不是將所有資料都完整地發送到數據中心。然後，我分析了實作這樣一個平台所必須面對的難題，並採用擁有龐大社群支持的開源專案來克服這些挑戰。最後，我對實作出的的平台進行全面的測試，結果顯示了：（i）分佈式分析的好處及其局限性，（ii）當跨多個運算裝置分發應用程式時，工作分配的重要性，以及（iii）平台中所使用的工具所帶來的額外運算成本。

Abstract

Modern Internet-of-Things (IoT) applications produce a large amount of data and require powerful analytics approaches, such as using Deep Learning to extract useful information. Existing IoT applications transmit the data to resource-rich data centers for analytics. However, it may congest networks, overload data centers, and increase security vulnerability. In my thesis, I implement a platform, which adopts the concept of Fog Computing, integrating resources from data centers (servers) to end devices (IoT devices). It has two features: (i)dynamic deployment and, (ii) edge analytics. I launch distributed analytics applications among the devices to pre-process the data, rather than sending everything to the data centers. I analyze the challenges to implement such a platform and carefully adopt popular open-source projects to overcome the challenges. I then conduct comprehensive experiments on the implemented platform. The results show: (i) the benefits/limitations of distributed analytics, (ii) the importance of decisions on distributing an application across multiple devices, and (iii) the overhead caused by different components in my platform.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet of Things (IoT) is getting popular all over the world. There are 6.4 billion IoT devices deployed in 2016, 30% more compared to 2015, and the number is expected to increase to 20.8 billion in 2020 [6]. Currently, the devices are used to sense/collect data and send them to powerful servers, such as cloud servers through the Internet for processing and further analysis. The analyzed results are then sent to IoT users for various IoT applications, such as smart cities, smart homes, and smart hospitals.

The incredible growth of the number of IoT devices results in severe network congestion and surging server loads under the huge amount of incoming data streams generated by many IoT devices. To solve these problems, we may try to pre-process the data using the IoT devices to extract some *higher-level features* for reducing the data size. Then send the reduced data to the server for analysis.

For example, assume there are a few IoT devices, and one of them captures an image. By sending this image directly to the data center, the size of data will be huge. But if pre-processing the image by the other devices on local to get the result first, such like the number of people or the plate of a car in this picture, then the size of data will be reduced drastically. In other words, the pre-processing idea will reduce the network traffic and the load of data centers significantly.

Therefore, analyzing those sensor data and extracting useful information on IoT devices is critical. Nowadays, such large amount of data is usually analyzed by Machine Learning (ML) approaches, especially using Deep Learning (DL). However, DL is too complicated for a single IoT device. Hence, a platform supporting distributed analytics is required to solve the aforementioned issues. We adopt *fog computing* [23] , which integrates resources from data centers to end devices to run applications in a distributed manner. That is, in fog computing platforms, we leverage resources from data centers, edge networks, and end devices. For brevity, we call these devices as *fog devices*.

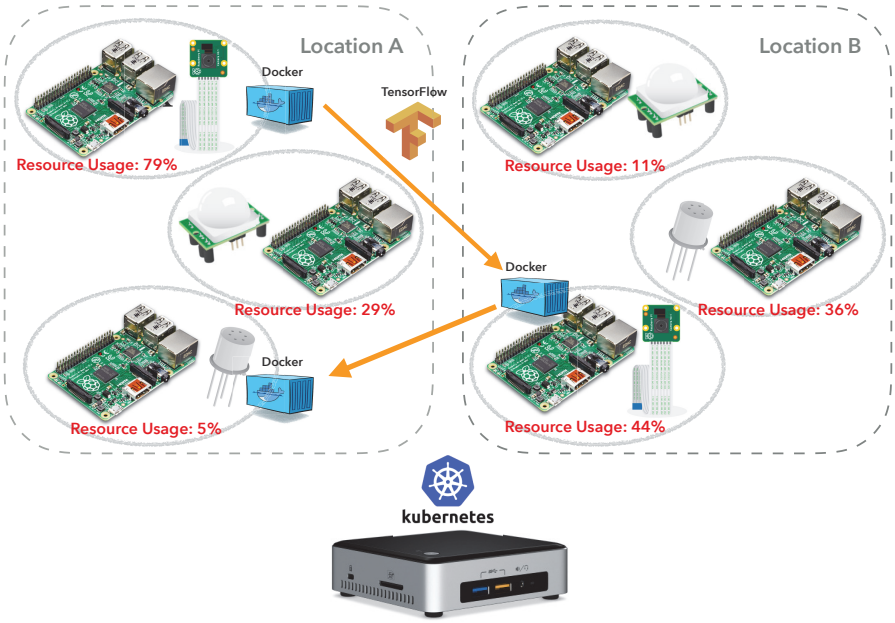


Figure 1.1: Overview of my fog computing platform.

1.1 Challenges

In cloud computing, devices refer to virtual or physical machines that have a lot of computing, memory and data resources. Devices carry out the actual workloads. Devices in the cloud usually hide in faraway data centers, which are vulnerable to insufficient bandwidth, long latency, and high product cost rate. Hence, cloud computing is less friendly to multimedia applications or the applications in the intermittent Internet. Recently, fog computing paradigm pushes devices toward edge networks. I adopt a generalized definition, which devices may be owned by the crowds. Such fog paradigm provides the resources closer to the users and reduces the cost. Unfortunately, managing the devices in the fog is challenging, e.g., end devices may move into wireless dead-zones, run out of battery, or be turned off. These factors lead to severe uncertainty. In this thesis, two major challenges are focused upon:

- **Heterogeneous devices.** There are many kinds of fog devices, such as desktops, laptops, and smartphones owned by different fog workers. This results in high heterogeneity, so implementing a fog computing platform that supports heterogeneous devices is really challenging because of the actual configurations of the devices and inter-connected networks need to be carefully optimized. Furthermore, fog devices on the edge usually have limited resources, thus I have to split applications into smaller *operators* running on the devices. Let them collaborate with one another to fulfill the complex job.

- **Large amount of data.** When the number of IoT devices is getting higher, the amount of data is incrementing. Because IoT devices usually have limited resources, sensor data tend to be sent to powerful data centers for processing. Sending large sensor data to the data centers may congest the networks and overload the data centers. The key to the solution is to collaborate several fog devices and pre-processing the data before transmitting them over the Internet. Therefore, how to analyze those sensor data and extract useful information is critical. Nowadays, such large amount of data is usually analyzed by Machine Learning (ML) approach, especially using Deep Learning (DL). However, DL is too complicated for a single fog device. Hence, a platform supporting distributed analytics is required to solve the aforementioned issues.

In this thesis, to overcome these challenges and implement a functional fog computing platform, I adopt three open-source projects: TensorFlow [18], Docker [4], and Kubernetes [9].

- **TensorFlow** is a library for numerical computations using data flow graphs. This idea of graphs suits the structure of distributed IoT devices; fog devices are nodes and the sensor data flows are edges in a graph. Hence, a cluster of weak devices can work together to perform resource-consuming analytics. Moreover, TensorFlow supports a wide spectrum of state-of-the-art DL approaches, which are useful to analyze a large amount of sensor data.

- **Docker** is a light-weight virtualization technology compared to traditional virtualization technologies, such as Xen [22] and KVM [10]. Although Docker cannot perfectly isolate virtualized applications, it is light enough to be used on the resource-limited IoT devices.

- **Kubernetes** is a state-of-the-art management tool of Docker. It plays two important roles in the tested fog computing platform for monitoring fog devices and deploying virtualized applications.

1.2 Contributions

In this thesis, I make the following contributions:

- I survey the existing problems of Internet of Things, which point that large amount of IoT devices and sensor data burdens the server and the network. So I propose a solution conducting the fog computing concept in IoT environment. Moreover,

I list and describe carefully two additional powerful features which fog computing can achieve In the end, I also study others research about IoT and fog computing and then compare them with my works.

- I implement a real fog computing platform. Figure 1.1 shows the big picture of this platform. Kubernetes monitors the information and resource status, such as CPU, memory and what sensor that fog devices have. Such information allows us to figure out which devices are more suitable to deploy the IoT applications. The applications are split into smaller pieces in TensorFlow, and these pieces are virtualized by Docker and deploy on multiple fog devices.

- I conduct extensive experiments to quantify the performance and limitations of the platform. My results show that: (i) running a complex analytics application in a distributed way achieves higher performance, (ii) deciding the splits of an application is critical to yield better performance, and (iii) the overhead caused by the container is rather low even on resource-limited devices.

1.3 Thesis Organization

In the rest of this thesis, I survey the current situation and challenges of the Internet of Things and discuss some possible solution in Chap. 2. After that, I propose a fog computing platform which can dynamically and scalably deploy the IoT analytics application. This platform has two major features: (i) dynamic deployment and (ii) edge analytics, which I will introduce in Chap. 3 and Chap. 4. Then I explain the overview of my fog computing platform in Chap. 5. Next it will be describing the tools I used to build the platform and giving some usage scenarios of the IoT applications in Chap. 6. In Chap. 7, I will implement the real fog computing platform and measure the performance of it. Lastly, the resulted works in Chap.8 and conclusions in Chap. 9.

Chapter 2

Background

In this chapter, I will survey the situation of current Internet of Things environment and conclude some challenges it encounters. Then I will analyze the fog computing which can solve IoT complications. In the end, I introduce the edge analytics which is the perfect implementation of IoT combined fog computing.

2.1 Internet of Things

The origin of the term ”Internet-of-Things” (IoT) can be traced back to 1999, which is first said by Kevin Ashton [24]. Originally, the Internet-of-Things was about using Radio Frequency Identification (RFID) to tag entities and connect them to the Internet. Their first obstacle was to expand the content of the Internet, because the data in the past were mostly produced by the human through recording, typing, capturing, and etc, making the types and the amount of information restricted. Therefore, if the things can automatically generate the data, the environment of the Internet would change significantly.

After a few years later, the concept of ”Internet-of-Things” become more completed [25, 40]. It not only covers the RFID technology, but also contains the wider field, such as sensors, actuators, Wi-Fi, 5G, LoRa, etc. With those technologies, the physical devices around our life can form a heterogeneous net, and all the entities in it can have their own unique identifications. Through these identifications and network technology, we will be able to control and manage them remotely. In addition, devices can exchange data themselves, to achieve the goal of automation.

The Internet-of-Things is becoming increasingly popular in our daily life. We see IoT applications everywhere, such as smart homes, smart factories, and smart cities. The ubiquitous IoT applications significantly change standard of living. For example, Echo [1] is a new IoT application manufactured by Amazon, which connects humans to other IoT devices via voice commands. With the growing IoT market, a forecast from Gartner says

that 8.4 billion IoT devices will be in use worldwide in 2017, 31% more than that in 2016, and the number will increase to 20.4 billion by 2020 [5]. However, this incredible number makes the development of the Internet of Things much challenging.

Along with the increase of IoT devices, the amount of data is also getting larger. Because IoT devices usually have limited resources, sensor data tend to be sent to powerful data centers for processing. However, sending large sensor data may congest the networks and overload the data centers. To solve this problem, collaborating several IoT devices and pre-processing the data before transmitting them over the Internet are the keys. For example, compared to sending large multimedia content, such as videos, to the data center for analysis, extracting the real meaning data on the IoT devices significantly reduces the amount of network traffic.

2.2 Fog Computing

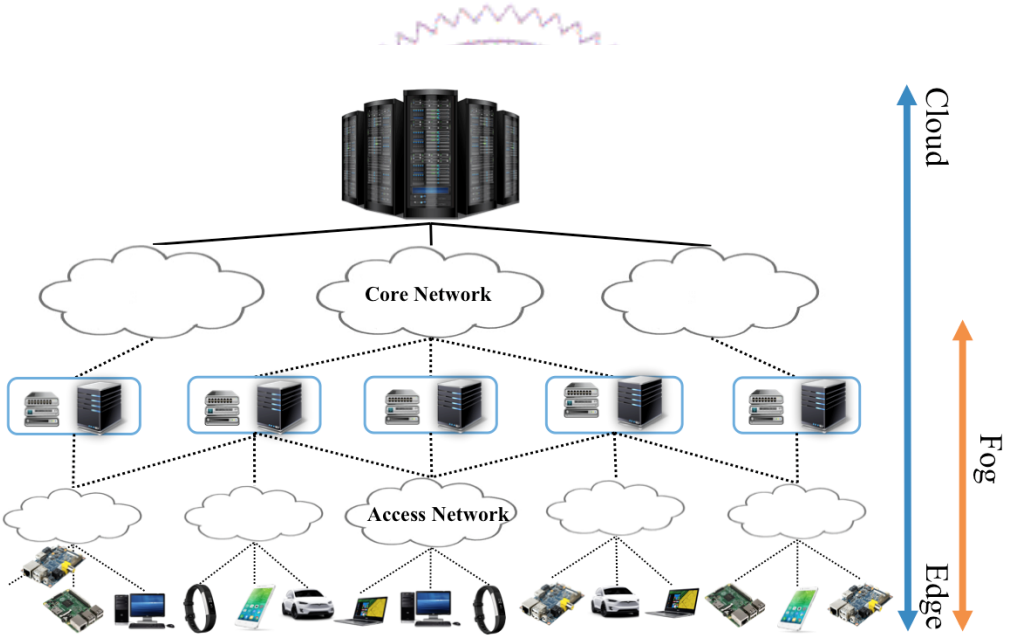


Figure 2.1: Architecture of the fog computing.

The fog computing concept is proposed and generalized by Vaquero et al. [26, 41], which has been employed for sensor-intensive applications [30] and computational-intensive applications [32]. Many researchers [28, 38, 43], thus, start to leverage the concept of fog computing to address IoT's issues. Fog computing leverages *fog devices* in data centers, edge networks, and end devices simultaneously in a distributed way.

Compared with the cloud computing, fog computing is proposed after the rise of the Internet of Things. As shown in Figure. 2.1., fog computing extend the cloud computing from the data center to the edge. Not only leveraging the powerful devices in the cloud but

also the less powerful *IoT devices* on the edge. fog devices can be the cars, refrigerators, street lamps, mobile devices or single-board computers which are diverse and scattered around the environment.

Because leveraging the edge side devices which are much closer to users than cloud side devices, fog computing can supply more kind of services near to the users. The advantages, such as unloading computing jobs from the data center to the edge network for low latency, location awareness, and adaption of the mobility and so on, are really crucial for IoT applications, especially when the type of sensed data is multimedia. For example, smart electric vehicles may take many pictures of the environment to determine the state of the road. Instead of sending the pictures to the cloud side, the car should process those pictures on the local side to achieve the goal of real-time response.

Unlike centralized cloud computing, fog computing is in the distributed architecture. It collects the computing resources that are closer to the user, and uses them to store or process the data, rather than pushing all the jobs to the cloud. Therefore, this also encourages users to share their idling resources of devices, including storage space and computing resources. The idea of outsourcing workloads to other fog devices can be viewed as the concept of crowdsourcing, which is benefits both parties. End device owners can efficiently utilize their idling resource and earn some remuneration. On the other hand, the users who outsource the workload can also finish the job at lower cost because they don't need to purchase the expensive server; and the negative externalities will decrease because of lower consumption of data center which results to less carbon footprint.

2.3 Edge Analytics

It is critical to analyze the sensor data and extract useful information on IoT devices. Nowadays, huge amount of data is usually analyzed by machine learning approaches, especially using deep learning. However, deep learning is too complicated for a single IoT device. Hence, using fog computing, which integrates resources from data centers to end devices, to support distributed analytics is required to solve the aforementioned issues. As a result, in fog computing architecture, we can leverage resources from several fog devices around the edge at the same time. It collects those resources to run complicated deep learning to pre-process and analyze the raw data before sending them to the cloud side.

Figure 2.2 is an example, in which we can split a complicated object detection application into three parts——image collector, features extractor, object classification——that goes onto three different fog devices. Let them cooperate with each other to finish the application. If we use the traditional way, sending the whole image to the data center di-

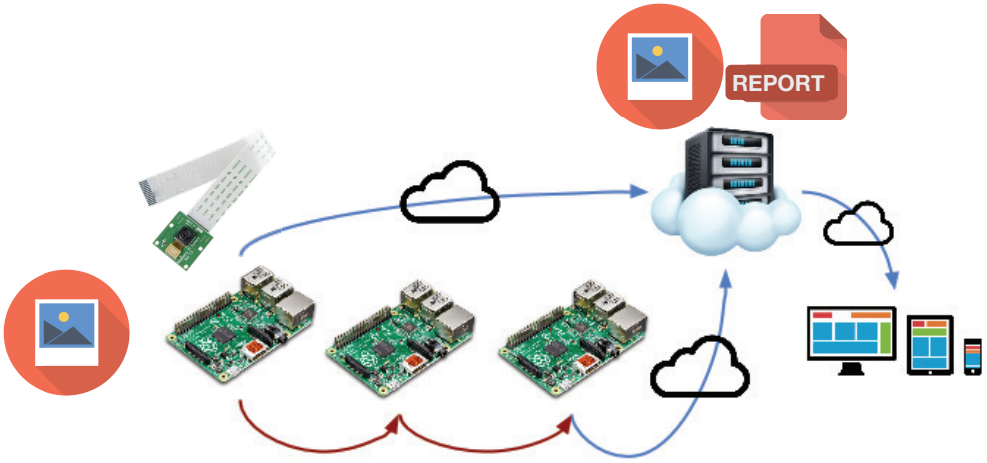


Figure 2.2: A graphical representation of a scenario that pre-processes the images before sending them to the cloud data center.

rectly, more network traffic is generated, and the response time may be longer. Therefore, following fog computing approach, pre-processing the images among the fog devices and send the concise reports to the data center will reduce the network traffic and the burden of data centers.

To implement a platform which using fog computing to support edge analytics, we have to overcome many challenges. First, we need a suitable programming model for distributed analytics. Second, because of the heterogeneity of fog devices, which have different capabilities, hardware, and software specifications, we need virtualization technologies to transparently deploy the IoT edge analyses applications. Third, to manage large numbers of IoT applications and fog devices, a centralized server in the headquarter is required to keep track of their status, and have the capabilities to dynamically deploy the applications on to those heterogeneous fog devices.

Chapter 3

Dynamic Deployment

IoT devices are prevalent in our daily lives and they usually are embedded systems. The embedded system is dedicated to specific tasks, the devices are specifically made for a particular goal. So the hardware on the system is usually fixed. Although such architecture can greatly optimize the execution performance and reduce the hardware size and production cost, it also leads to many problems: i) Limited flexibility of application and inability to handle more complex tasks, (ii) difficult management and monitoring needed for several embedded devices centrally, (iii) inefficiency when unmounting before conducting maintenance or update every time, which takes considerable labor costs and time, and (iv) inefficiency when embedded devices can not be re-use under different situations. In order to solve above problems, I developed a fog computing platform which dynamically and remotely deploy the applications onto any fog devices, so the users can launch whatever application whenever and any they want. Therefore, the software on the devices can also be updated or corrected in real time.

To achieve the goal of dynamic deployment, I subdivide this big challenge into couples of tasks. First, I need to virtualize the applications so the applications can be put on any heterogeneous devices directly. Thus, developers don't have to consider the setup of the running environment because all the setting already been packaged into the virtual machine. Second, as shown in figure 1.1, we need a centralized headquarter to remotely manage all the fog devices and applications running on them. From this I can monitor the resource usage and status of applications and fog devices and record the sensors that devices have and locations of the devices. After that, according to the information, I can dynamically deploy the applications which are packaged in the virtual machine onto the suitable devices. As a result, all fog devices can be managed centrally, which allows remote the maintenance.

Third, after implementing dynamic deployment, we also developed the event-driven mechanism that allows application developers to design a series of logical decisions that

automatically deploy another application to the right devices when a specific event is triggered. This increases the elasticity and ability when we deploy the applications on fog devices.

In the following subsections I will describe these three tasks in details.

3.1 Container-Based Virtualization

If all the applications are executed or updated directly on the same operating system, it has the risk of some conflicts due to library dependency, and the resources usage and permissions are also not easy to manage, even worse, the fails of the application also may cause the crash of operating system. In order to avoid this kind of situation, I separated each application that I planned to execute, and used virtualization technology to encapsulate them and put them into virtual machines. Each virtual machine is independent. They will not interfere with each other, so the developers can also limit their authority and available resources. In this way, the problematic situations mentioned previously can be solved smoothly.

Virtualization technology also brings several additional benefits. (i) Virtual machines can be started, stopped, or even migrated optionally, which improves the flexibility of various applications on the fog devices. (ii) The library, required by the application, will also be packaged in the virtual machine with the application, which means that library does not exist when the user doesn't have that virtual machine on fog devices. It saves lots of space, in particular, some large library such as OpenCV. (iii) The library required by the application can only be compiled once before being packaged. After that, the same virtual machine can be reused. It does not have to take another hour to recompile, which will save a lot of time.

Virtualization technologies, such as Xen [22] and KVM [10] need entire guest operating systems, the necessary binaries, and libraries to create a virtual machine. It needs powerful server to launch a virtual machine, which consumes a lot of storage space and computing power. These disadvantages make the traditional virtualization technology unsuitable for fog devices because fog devices usually have fewer computing resources. Using traditional virtualization technology, only less than ten virtual machines can be executed simultaneous on one fog devices with low efficiency. In addition, some IoT applications require real-time response, but traditional virtualization technologies need a long time to startup, which leads to a significant increase in time inefficiency. Therefore, I used a more lightweight virtualization technology in the difficult IoT environment.

Currently, an emerging virtualization technology called containers, such as LXC [12] and Docker [4], is used in several areas. Compared to the virtual machine, container

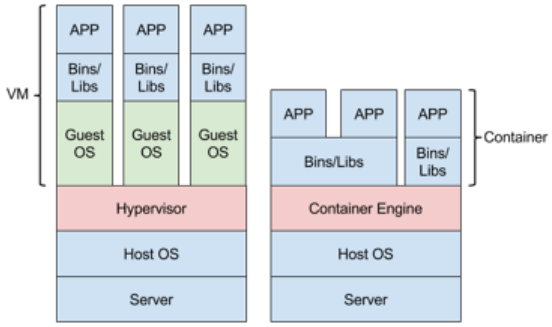


Figure 3.1: Difference of architecture between container and traditional virtual machine.

requires fewer resources and can be set-up in a short time span, due to its architecture containers omit the guest operating systems. As shown in figure 3.1, multiple containers can run on the same host, share the same operating system kernel and use the namespaces to distinguish one from another. Rather than including the full operating system, It allows applications running in containers with mandatory services they need. This design gets rid of the non-essential library, leaving only the library that applications need. Therefore, the resource required for building a container is relatively less, and the required computing resources are almost the same as bare metal. This allows the container to be boot up in a really short time even in a resource-constrained environment. As summarized in Table 3.1, these advantages surpasses the traditional virtualization technology and are the perfect choice in the IoT environment.

Table 3.1: The comparisons of efficiency between container and traditional virtual machine

	Container	**Traditional Virtual Machine**
Start Up	Seconds	Minutes
Capacity	MB	GB
Efficacy	Almost native	Slow
Scalability	Thousands	Dozen

3.2 Orchestration Tool

A premise of dynamic deployment is that we must know which fog device has enough computing resources to execute the application we want. In addition, the platform must also have the authority to control the fog devices, so it can assign the applications onto any devices directly. Therefore, a central headquarter to monitor various resources of the devices, including the CPU and the memory, and push the applications, packaged into the

containers mentioned in the previous section onto the suitable devices. However, there are heterogeneous fog devices in the cluster of the fog computing platform, and the network interfaces conducted by the devices are usually not the same. Therefore, managing them using the same method will be challenging. Furthermore, the applications are packaged by virtualization technology, so extracting the information inside the containers and maintaining the current status is another problem. When the application or container fail, it should be corrected in the short time. But it will take a lot of time if it needs to wait for the manual modification. This kind of circumstance is not allowed to happen especially at the factory or hospital. Therefore, only if we dynamically deploy the applications onto the right fog devices, the platform should have the ability to automatically repair and correct the applications.

To achieve the goal I mentioned, the fog computing platform needs an orchestration tool to monitor and control all the devices and containers. SaltStack [16], Docker Swarm [17], OpenStack [15], and Kubernetes [9] are all possible open-source projects as the starting point to implement my fog computing platform. Following I will introduce them briefly.

- **SaltStack** is lightweight management software written in Python to remotely configure and execute parallel commands on many computers. SaltStack adopts a master/minion architecture. Master is a centralized server to control all the end devices as known as minions. SaltStack supports multiple operating systems and has been integrated with Docker [4].

- **Swarm** is a native clustering system for Docker. This makes Swarm simpler for the developer to learn and use, thus reducing the cost of development. Swarm assembles multiple containers into a cluster which provides the services. It uses the standard Docker API to manage the life cycle of the container to schedule the container. Swarm has two major component: Manager and Node, which is similar to the master/minion architecture.

- **OpenStack** is one of the most widely used cloud management systems. It helps cloud service providers to efficiently manage their large pools of machines in data centers. OpenStack offers management services for virtual machines (Nova), images (Glance), networking (Neutron), storage (Swift and Cinder), and security (Keystone). Moreover, OpenStack provides Web-based interface (Horizon) to control all the services. OpenStack supports different virtualization technologies, including Xen [22], KVM [10], and LXC [12].

- **Kubernetes** is a framework proposed by Google to manage a cluster of Docker containers. Kubernetes also consists of a master and many minions, but it is more

comprehensive than SaltStack, e.g., Kubernetes helps users to ensure redundancy for better fault tolerance. Each minion can hosts several containers, and these containers can gather together and form the base unit of Kubernetes, termed pod. Furthermore, Kubernetes has many different types of controllers to assist developers to manage the pods in the flexible way, such as making sure there is at least one certain pod running in the cluster; if one of the containers failed, the Kubernetes will automatically reboot another one. This feature is really useful for automatic reparation and correction.

I selected Kubernetes to implement the fog computing platform because it supports Docker, the latest and the most popular light-weight virtual machine. It follows a master/slave architecture, and slaves don't need to install too many packages on it which is suitable for fog devices. Furthermore, this project is developing rapidly, many extensions have been implemented for Kubernetes. And its various and powerful controller also allows developers to manage their containers more efficiently. For those reasons, Kubernetes is my best choice for implementing the platform.

3.3 Event-Driven Mechanism

The former two sections fulfilled the capability of the dynamic deployment mechanism of my fog computing platform. But what kind of applications should be deployed on fog devices and before the deployment what is the minimum amount of resource will manually designated by the users every time. Therefore, the platform can not automatically deploy another application when an urgent and unpredictable situation occurs. For example, when something is detected by the passive infrared (PIR) motion sensor of the security system, the platform should automatically trigger another application to recognize the object and distinguish whether it is an intruder and whether to notify the relevant people. Of course, making this application always run in the background is also a possible solution. However, if most of the time no one passes through this particular space, then the object detection application is mostly idle and wastes substantial computing resources that could be used for alternatives. Therefore, in this scenario, we need to develop an event-driven mechanism so that the application, triggered manually at a certain point, can be automatically deployed to an appropriate fog device when any event happens.

There are two different way to implement an event-driven mechanism: (i) all the event-driven rules are managed in one single decision unit, or (ii) the first application drive the next application autonomously. The first method is able to centrally manage all the event-driven rules and establish the rules intuitively in one-to-one correspondence. In other words, the corresponding B application is launched directly when the event A

occurs. This method is simple because there is a standard template for developers to follow; however, on the contrary, because of it, it is relatively rigid and non-flexible. And the second method is to trust each application to decide when to drive the corresponding application. The logic of the decisions are made in the first application, so developers are not constrained to any template of the rules. They can establish the rules as complex as they want. Both methods have their advantages. On my platform, in order to maintain the high flexibility of the IoT environment, the second method is adopted. Therefore, an API-Server which allows the application communicate with the headquarter is crucial. In other words, this API-Server provides IoT applications in the fog computing platform a way to send the request to headquarter for some manipulations such as deploy an application on another device.

Chapter 4

Edge Analytics

In this chapter, I'm going to describe another feature of my fog computing platform: edge analytics. Edge analytics enables IoT devices process the complex deep learn application on the edge. Pre-processing the raw sensor data before transmitting them to the cloud server, so that reducing the burden of data center and even shortening the response time.

Processing deep learning application usually consumes lots of computing resource, so most of the fog computing system dispatch this job to the cloud side. Under that kind of architecture, all the raw data needs to be sent to the server, which is far away from the users, to not only occupying much bandwidth, but also lengthening the response time. In order to solve these problems, I made IoT devices, which use deep learning to support analytics applications. Futhermore, in order to distribute the workload, I conduct distributed architecture which split one complex analytic application into several ***operators***.

There are many distributed computing studies, and some of these studies focus on collecting computing resources from volunteers around the world, which are relative to fog computing. By effectively using the idle devices such like desktops, laptops, or smartphones to do computations, even if these devices are far less capable than a powerful server. Futhermore, they can still accomplish quite a few tasks after they cooperate with each other. Not only does it avoid the waste of idle resources, but also eliminates the cost of building the servers. Such concept can also be applied to my platform. By combining multiple IoT devices, I am able to effectively use the remaining idle computing resources to process some complex analytic applications and relief the workload of the cloud servers.

In the following sections, I will introduce the distributed computing in my platform, and describe how it adapts with the IoT applications which are location-based and sensor-based. In the end, I will explain how resource-constrained IoT devices handle power-consuming analytic application in the distributed way.

4.1 Location-based and Sensor-based Services for the Internet of Things

Different from the general distributed computing. I run the IoT analytic applications on the decentralized IoT devices, which derives many limitations. The types of IoT devices are often different and scattered around the environment, and the applications are mostly location-based services. Therefore, it is necessary to select the IoT device located in the correct geographical position when executing the application. In addition, different applications require different sensors. It is import to consider the suitable sensor each device is equipped with. For example, when we need to deploy a license plate recognition application, we would like to select an IoT device that is located at the side of the road rather than inside the building. The device should akso be equipped with a sensor with camera function rather than a microphone function so that the application can complete its work.

Some of the IoT devices are not fixed at the fixed position, such as the devices that are installed on a bus or carried around by the users. Sensors on the devices may also change due to various factors. Therefore, this information will not be the same forever. So the issue is: how the fog computing platform dynamically get the current metadata of these devices so the platform can deploy the applications correctly. In addition, among a large number of IoT devices another challenge is how to quickly and efficiently filter out devices that meet the requirements.

In order to confirm whether the IoT device meets the location and sensor requirements of the application. The most practical method is to tag the IoT devices with various labels and keep this data on the master, which allows the deployment of applications on the right IoT devices base on this information. Therefore, the devices can be classified in a simple and intuitive way, and even find out devices that satisfy multiple conditions at the same time directly and logically. In order to solve the problem of outdated information, each IoT device also needs to be able to update the label independently to the master for maintaining the correctness of the information.

4.2 Distributed Computing

Many existing distributed computing studies focus on splitting an amount of data into several parts which are similar. For example, separating one video into multiple frames or dividing thousands of data into ten equal batches, and then processing each part by different device in the same way. Then the results from the different devices will all be gathered together at the same cloud server, which the server will aggregate them and get

the final conclusion. In this method, there is not much interaction between the devices and the server, which is more suitable for the applications with high computational complexity and a abundance repetitive data. However, in the IoT environment, the diversity of the data is high, so it is difficult to integrate them. In addition, the IoT sensor data usually are time-sensitive and have to be processed in real-time. Therefore, it is critical to allocate different jobs according to the characteristics of heterogeneous devices and get the results in a short time. For example, the license plate recognition application can be divided into two tasks: taking pictures and recognizing the numbers of license plate. The first job is performed by a device equipped with a camera, and the second job is performed by a device with more computing resources. In this way, it can effectively use the advantages of each device to improve the computing efficiency.

If an application is divided into multiple operators based on the characteristics of different jobs, and assigning them to different devices, this means that a pipelined execution structure is formed. The previous operator will calculate a preliminary result, and then transmitting it to the next operator. Therefore, this also leads to two problems: (i) where are the cutting points of the applications, and (ii) how the operator and operator communicate with each other?

The result of each operator must yield a productive result, such as the temperature, an image, the name of the object on the picture, etc. The advantage of doing so is that the application can be disassembled, modularized, and used effectively. For example, both the garden irrigation system and the weather prediction service can obtain data from the same operator that senses the humidity. Thus, we can save resources by only deploying a humidity operator.

As mentioned in the previous paragraph, there are situations that multiple operators need the data from the same operator at the same time. Therefore, transferring the data one-to-one is inefficient. In order to solve this problem, we adopted publish/subscribe communication model. All the results of the operators are published to the broker. In other word, the operators that need these data subscribe them from the broker themselves. Futhermore, because the data are all sent and received through the broker, between the operators, they do not need to build their own communication channels; the complexity of developing the applications can be reduced.

4.3 Pre-processing with Deep Learning

It is time-consuming to run the edge analytic application to pre-process the raw data, which uses complex deep learning, on the resource-constrained IoT devices. If we further cut the edge analytic application into multiple operators properly, then deploy them on

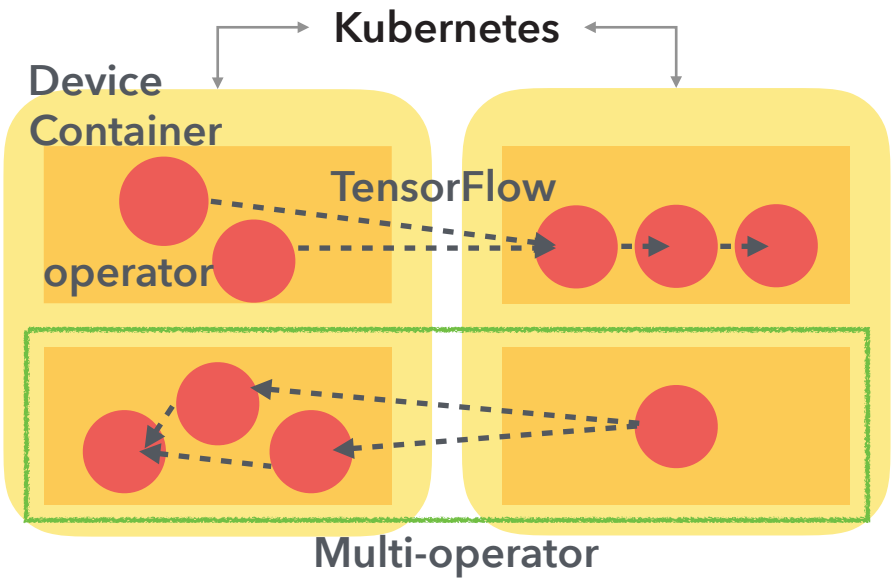


Figure 4.1: The architecture of my fog computing platform.

several devices to run the application, it will get better efficiency comparing to run the whole application on only one device. Continuing our license plate recognition example in Sec.4.2, this section further splits the second job, plate recognizing, into parts, based on the layers of the neural network, variables, or the functions. Throug this, we then delivering them to multiple IoT devices for processing, so that complex deep learning can also be executed well on the resource-constrained IoT devices.

The difficulties of pre-processing in distributed system with deep learning is similar to Sec.4.2: (i) where are the cutting points of the applications, and (ii) how the operator and operator communicate with each other. Even so, the differences of cutting degree(a complete job versus one single neural network layer) makes it a more difficult challenge. The library used for deep learning must be able to separate the complete neural network into multiple layers, and support these layers so can be processed on heterogeneous IoT devices. Although the heterogeneous IoT devices problem can be solved by virtualization technology, the communication between layers cannot be solved by pub/sub model. So the deep learning library should handle the communication issue itself.

TensorFlow is a library for numerical computations using data flow graphs, originally developed by Google's Machine Intelligence research organization. It has a very an extremely flexible architecture that can be implemented on desktops, servers, or mobile devices. TensorFlow helps developers build data flow graphs by breaking up the application into smaller operators. These operators are then deployed on decentralized IoT devices, which allow them to use Multi-Dimensional Data Arrays, or tensors, to communicate with another device. As shown in Figure 4.1, we can combine the results of the Chapter 3 to package the operators generated by TensorFlow using container virtualization technology,

and then deploy them to the appropriate IoT devices through Kubernetes.

Chapter 5

System Overview

Figure. 5.1 gives an overview on my fog computing platform, which consists of the following major components.

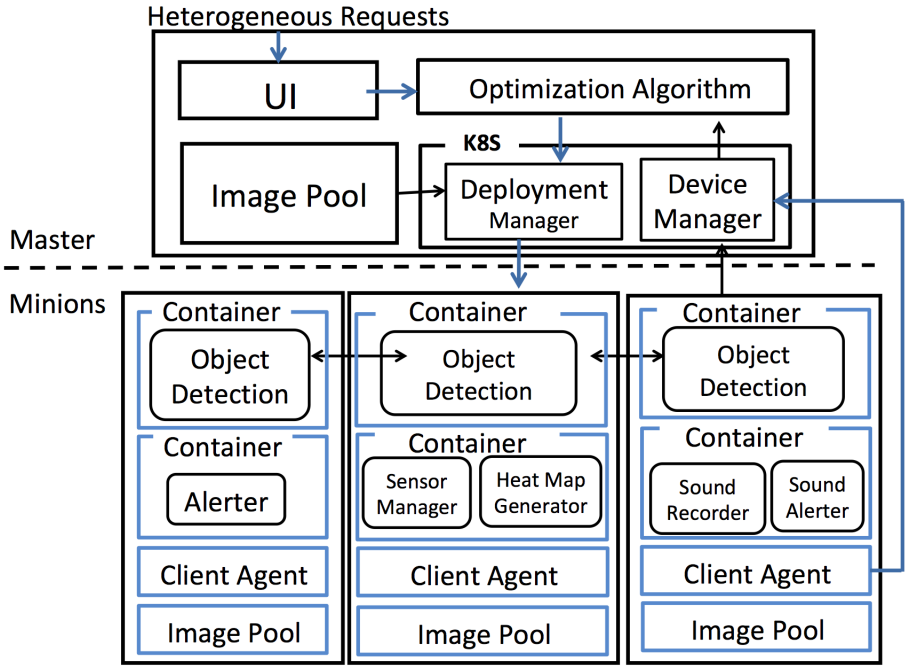


Figure 5.1: The fog computing platform proposed for IoT analytics.

5.1 Master

- **Device manager.** In the fog computing cluster, we need to monitor the status of every fog device (minions), the data such as: CPU, RAM usage etc. We extend the Heapster and cAdvisor in Kubernetes [9] to collect crucial device status, including: (i) the resource utilization, e.g., CPU and RAM usage, (ii) the device locations, e.g., GPS readings, and (iii) sensor availability and capability, e.g., existence of depth

cameras at VGA resolution. These information not only enable user to understand the situation of cluster, but also allow optimization algorithm component compute the best strategy to deploy the applications.

- **Image Pool.** Application developers package each operator of IoT analytic application into a Docker image. These images may be stored on the online servers, such as Docker Hub, or in image pool on the master or fog devices to avoid downloading duplicated images. Image pool also enables version control, and leverages the dependency among Docker images to further reduce the resource consumption.

- **Optimization algorithm.** For all IoT analytics requests, the algorithm makes decisions that deploy which operators on which fog devices. On my platform, the algorithm can be replaced on-demand. Users can switch the algorithms to which satisfies their usage scenario. Such as the one processes maximum number of the requests, the one finishes the requests fastest, or the one can sort the priorities of requests, and etc.

  In this thesis, the requests come with diverse QoS targets and deployed locations. As shown in Figure. 5.2, the goal of the algorithm is to maximize the number of satisfied requests, while considering: (i) the expected resource consumptions (system models), (ii) the spatial requirements of requests, (iii) the required sensors, and (iv) the available resources of fog devices. A request is satisfied if the target QoS is achieved at its specified location. Due to the space limitations, the details are given in our paper [31].

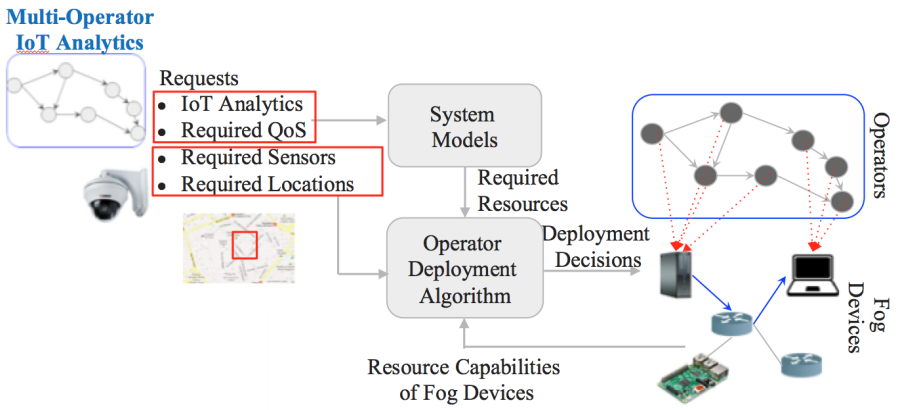
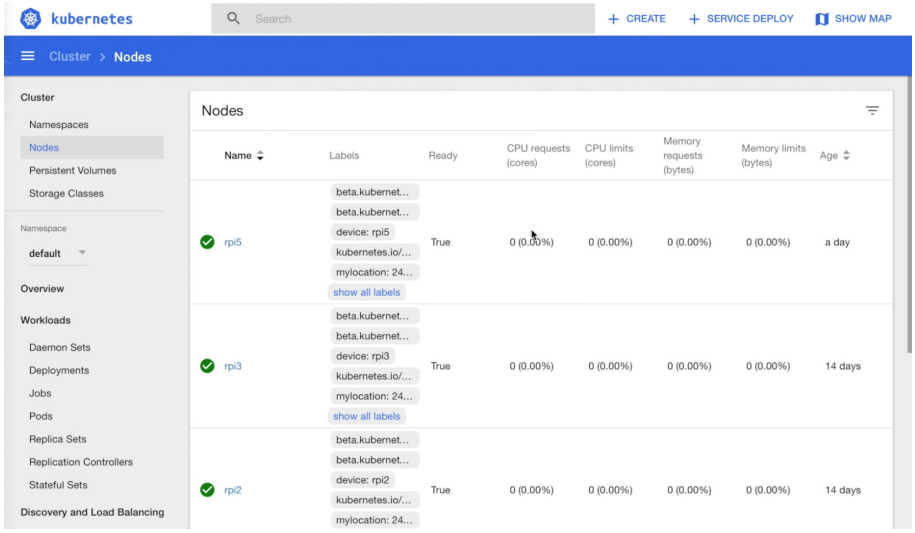
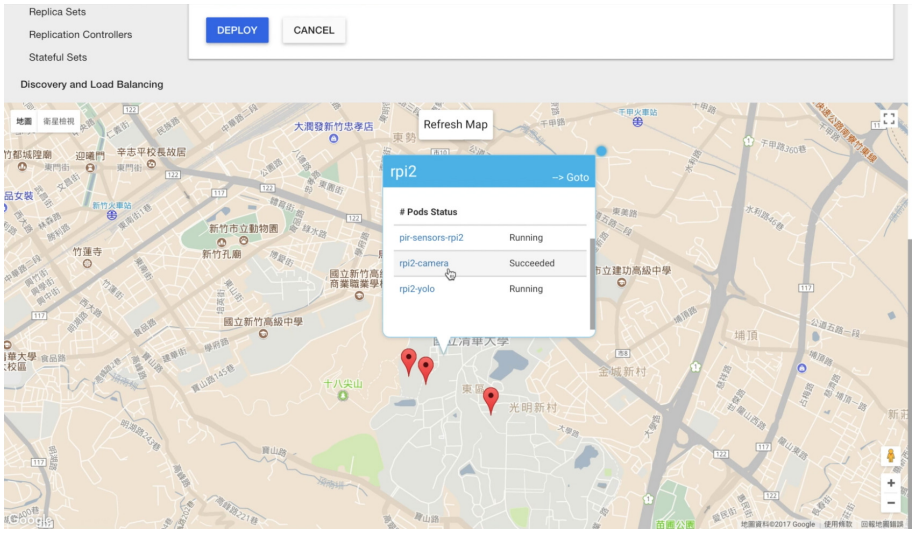


Figure 5.2: The overview of our operator deployment algorithm.

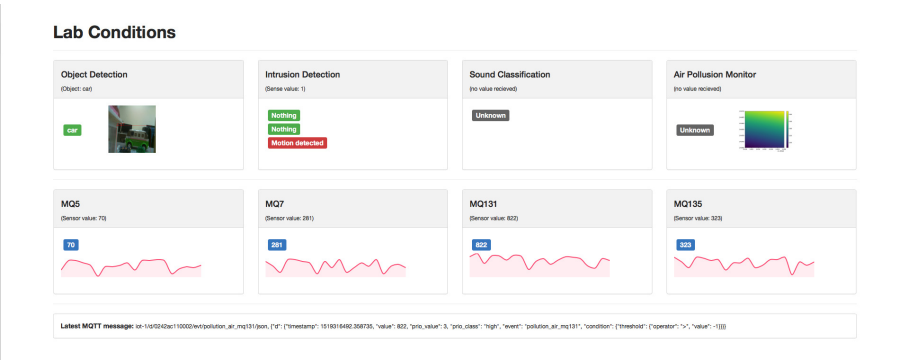
- **Deployment manager.** We build a deployment manager on Kubernetes to remotely launch specific Docker images on chosen fog devices, and support other management tasks, such as killing, restarting, and migrating containers.



(a)



(b)



(c)

Figure 5.3: User Interface

- **User interface.** As illustrated in Figure. 5.3, we also provide the dashboards for the user to operate or monitor the fog computing platform conveniently.

The open source project Kubernetes dashboard is an extension of the Kubernetes, which allows the users using the user interface to manipulate Kubernetes API directly. Therefore, we can check the status of every node or container on the platform, and deploy the application in any methods Kubernetes provided. However in IoT environment, we need more features. So I extend the Kubernetes dashboard, add a map to visualize the positions of every device. As Figure. 5.3(b) shows, when fog devices scattered around the city, we can easily find the fog devices on the map through their locations, rather than searching by latitude and longitude. Furthermore, infowindows of the markers also display what sensors that fog devices have and the status of the applications that run on the fog devices. These information help users to decide which fog device is suitable for their requirements more efficiently.

I also implemented a panel to display the results which is subscribed from the analytics applications through MQTT. As Figure. 5.3(c) shows, the panel visualizes the information, so the users can read the data efficiently. Such as which gate is being intruded, the picture of the intrusion scene, or the motion chart of the air pollution values. This panel gathers the results from different analytics applications and put them on one single page, so the users can check all the results directly and efficiently.

5.2 Minions

- **TensorFlow-enabled container.** Containers are light-weight virtual machines suitable for IoT analytics, because of their smaller footprints and shorter initialization time. We include TensorFlow and its analytics libraries in Docker containers.

- **Client Agent.** An agent that runs on each minion in the cluster. It makes sure that containers are running in a pod. And also report the resource usage of minion and the containers running on it to device manager periodically. The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are functioning healthily. The kubelet doesn't manage containers that were not created by Kubernetes.

Chapter 6

Implementation and Demo Scenarios

6.1 Implementation

In this thesis, I adopted TensorFlow [18] as my programming model which is responsible for edge analytics, Docker [4] as my virtualization tool, and Kubernetes [9] as my orchestration tool. Docker and Kubernetes——are responsible for dynamic deployment, which I will describe in detail in the following.

6.1.1 TensorFlow

TensorFlow helps us build the data flow graph by splitting an application into multiple smaller *operators*, which are the units of deployments. TensorFlow deploys those operators on distributed fog devices and allows them to communicate with one another using multi-dimensional data arrays, called tensors. TensorFlow has a flexible architecture that can deploy applications on desktops, servers, or mobile devices with the same API. My fog platform leverages on that and deploys various operators on heterogeneous fog devices.

With TensorFlow, each applications can be written as a graph of multiple operators, which has several advantages, compared to the single-operator ones. First, they allow fog devices to only perform the jobs they are good at, which solve the problem that fog devices are not capable of due to resource limit. Second, multi-operator applications can save the resources by reducing the amount data transferred. For example, if there are two operators on different fog devices needing for same input data, they can get the data from the same operator, and reuse the operator rather than collecting data separately. Last, running TensorFlow applications to pre-processing the raw data on the fog devices reduces the burden of networks and servers. IoT applications usually transmit the data to the cloud side, but the incredible amount of data could congest the network and overload

the server. Multi-operator applications can perform pre-processing, such as filtering or merging when the data goes through the operators, to yield results at the edge to reduce the size of data that are transmitted to the server.

Figure 2.2 is an example, in which we split a crowdedness-detection application into three operators: image collector, face detector, crowdedness monitor. If we use the traditional way, more network traffic is generated, because send the whole image to the data center. Following my fog computing approach, we pre-process the images among the fog devices and send concise reports to the data center, which reduces the network traffic.

6.1.2 Docker

Each operator may require different libraries, but we can't install all the libraries on every fog device, which takes too much time and storage. For example, some multimedia libraries, such as OpenCV, is large for fog devices. Therefore, my platform adopts Docker containers to package the operators with their required libraries into Docker images. Hence, we do not need to install libraries on my fog devices. The operators and libraries are packaged as a container for on-demand deployment.

Similar to traditional virtualization technologies, container technology is the solution to transparently deploy tasks on heterogeneous devices. However, container technology is lighter and more scalable. It is suitable for fog devices which don't have enough computing resources. Besides, the short startup time is also helpful for real-time IoT applications and fast deployments. Also, if developers want to change some algorithms of an application or the configuration of operator graphs, they can easily launch a whole new container, since the overhead of restarting a container is small.

6.1.3 Kubernetes

For those containers running on heterogeneous devices at different locations, we need a tool to manage and monitor them, such as Kubernetes [9], OpenStack [15], Docker Swarm [17], and SaltStack [16]. We selected Kubernetes because of its popularity. We gain full control over all the operators in every application using Kubernetes. It has an extension, named Heapster, which enables container cluster status monitoring, resource monitoring and error recovery. For example, when a fog device runs out of batteries or is disconnected from the Internet, Kubernetes can automatically relaunch the new containers on another fog device to keep serving our applications.

As illustrated in Figure 4.1, Kubernetes plays the role of the centralized server in the headquarter. It monitors the status of fog devices and deploys virtualized operators on them according to the monitored information. Every fog device runs multiple containers,

which are part of different applications. If one of the applications needs to be modified, we simply replace that container, which avoids interrupting other applications.

6.2 Scenarios

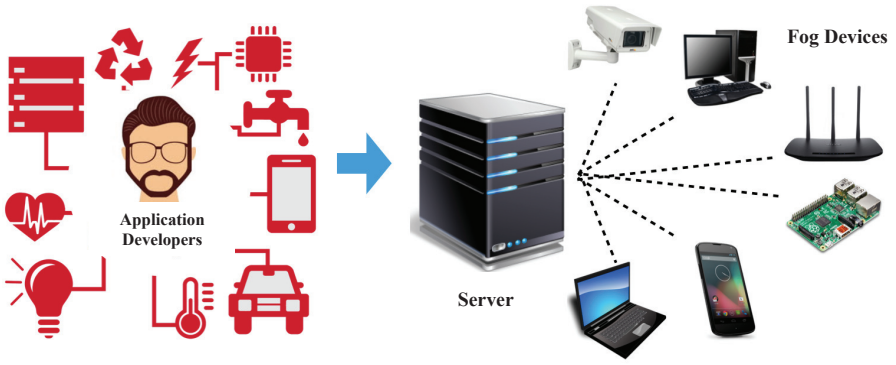


Figure 6.1: The ecosystem of my fog computing.

In the thesis, we envision an ecosystem that consists of the centralized server, fog devices, and the applications which are developed by developers. Developers use Docker and TensorFlow to develop the applications which can process the distributed analytics. Upon getting the requests, the centralized server deploys the applications onto heterogeneous fog devices through Kubernetes. As shown in Figure 6.1, this structure build trust about the details of the deployment, so they can focus on the developing of the applications.

I implement the multi-operator IoT analytics using TensorFlow [18], in which every application is represented as a graph consisting of *nodes* and *links*. The nodes can be simple tasks, such as additions and subtractions, or very complex machine learning tasks. The links are the data flows among nodes. I use TensorFlow *tags* to split every graph into multiple smaller subgraphs, where each subgraph contains one or multiple nodes and associated links. These subgraphs are essentially the *operators* in our fog computing platform, which are the units of deployments.

However, IoT analytics may require some features that are not available in TensorFlow. For example, OpenCV [14] is needed for processing images and librosa [11] is needed for analyzing audios. These additional libraries are quite large and take hours to compile. Therefore, installing these libraries on-the-fly is infeasible. On the other hand, reimplementing the features in TensorFlow incurs high engineering overhead, which may

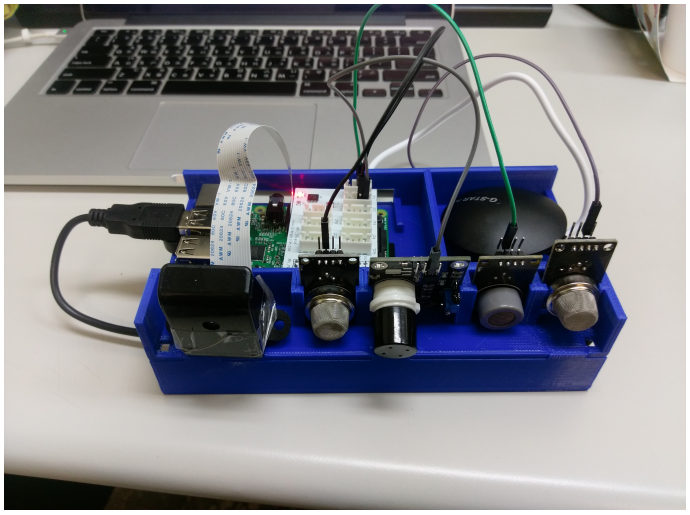


Figure 6.2: The SCALE box with various sensors installed.

not be worth it. Fortunately, our fog computing platform achieves short deployment time using Docker images. Furthermore, TensorFlow does not natively support stream processing, and we add a queue between any two adjacent operators to increase the overall performance.

SCALE [39] (Safe Community and Alerting Network) is a project that aims to use available, cheap sensors to build an affordable, practical security system. The sensors could be motion detection sensors, gas sensors, temperature sensors, or heartbeat sensors. The sensor data are published via the *lightweight* messaging protocol MQTT [13] to a broker. Devices can get those data by subscribing them from the broker. The devices then pre-process those raw sensor data to get meaningful results. We enhance this project to support two crucial sensors: camera and microphone. We then put the Raspberry Pi which installed the SCALE in a 3D-printed box (illustrated in Figure 6.2) and putting it at a fixed position such like streetlight. Another way is to attach the boxes onto bikes and collect sensor data along the path where the biker passes. In the rest of this section, we present three applications based on SCALE.

6.2.1 Object Detection

You Only Look Once (YOLO) [35] is a real-time object detection application. It applies a neural network to an entire image, and performs both classification and localization. The recognized objects are marked by *bounding boxes*. This network divides the image into multiple grid cells and predicts the bounding boxes and the probabilities for each cell. In the end, it marks the object as a category that has the highest score. We pre-train a model that can detect dogs, buses, motorcycles, human beings, and etc. We use this model to analyze the images that are captured by the camera on SCALE box. Figure 6.3 shows the

Figure 6.3: A sample result from YOLO.

results we yielded from an image.

There are two ways to distribute jobs of YOLO. The first one is distributing an image to multiple nodes, which concurrently analyzes different grids of the same image. But the YOLO application needs OpenCV library, which is too large to be installed on all nodes. Therefore, in this thesis, we use an alternative way: we split YOLO into multiple operators, one of them is responsible for the OpenCV part, and the others are responsible for TensorFlow's built-in DL algorithms. Thus, every image will be pre-processed by multiple operators among multiple devices as a distributed pipeline. Doing so avoids the situation of filling up the storage space with libraries and it even increases its efficiency if we put the operators on the right devices.

6.2.2 Sound Classification

Urban sound classifier application implements Multilayer Neural Networks using TensorFlow. It can classify the sound recorded by the SCALE box. I already trained a model which can determine 10 classes [19]: air conditioner, car horn playing children, dog barks, drilling, idle engines , gunshots, jackhammers, sirens, and street music. The classes are drawn from the urban sound taxonomy. We can use these results to do many things, such as notifying the police when the microphone records the sound of a gunshot, or marking the address as a potential construction project when the microphone records the sound of jackhammer and drilling. We split this application into multiple operators according to its

Multilayer Neural Networks, every operator is responsible for one layer. Similar with the YOLO application, distribute layers are more storage-efficient, and faster if we put the operators on the right devices.

6.2.3 Air Quality Monitor

Air Pollution Detection subscribes the sensor data from gas sensors and calculates the moving average values of each type of sensor data, such as natural gas, carbon monoxide, or smoke. When the average value exceeds a threshold, which implies the air quality is harmful to health, then we will alert the citizens or inform the government. There are quite a few gas sensors working at the same time, and the amount of sensor data is non-trivial. Therefore, this application needs to classify the data type and distribute them to multiple operators first. Every operator is responsible for one sensor and calculates the average value of it. My fog computing platform will put those operators on different fog devices, which lowers the risk of excessive burden is incurred on a single device.

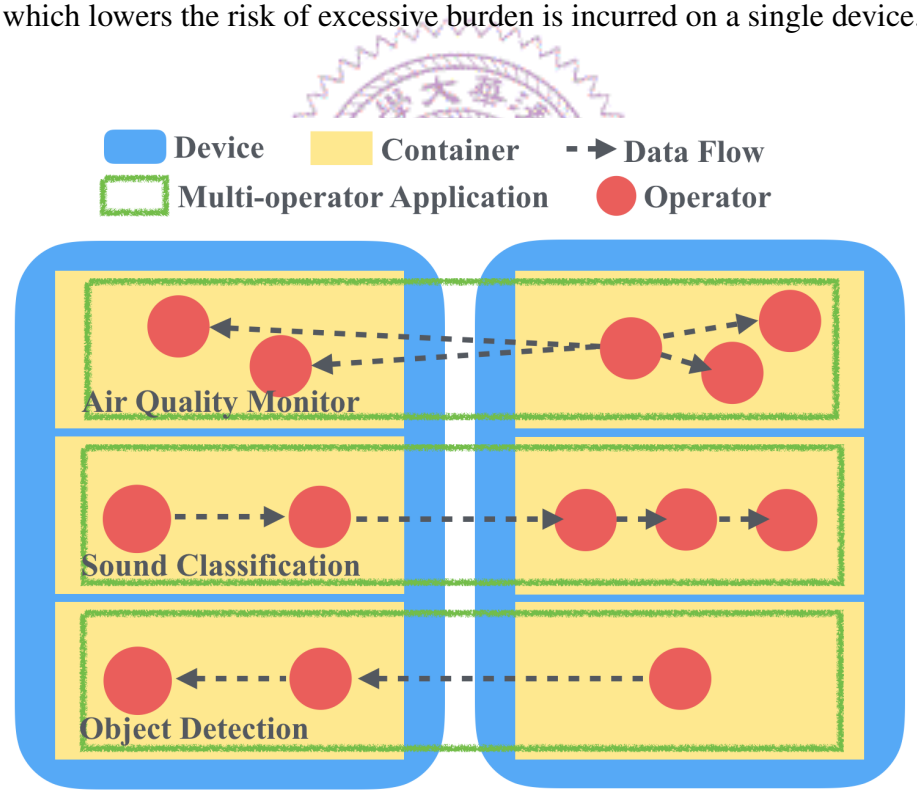


Figure 6.4: Three different edge analytic applications.

As shown in Figure 6.4, I already introduced the applications that I run on my fog computing platform. As you can see, it contains the different type of data: string, image, and audio. And each application has their own features and challenges. In the next section, I will measure the resource usage of those applications, which run in containers and evaluate the performance of my platform, to determine if the structure is suitable for

distributed analytic.

Chapter 7

Evaluations

In this chapter, I conducted real experiments to evaluate my fog computing platform.

7.1 Setup

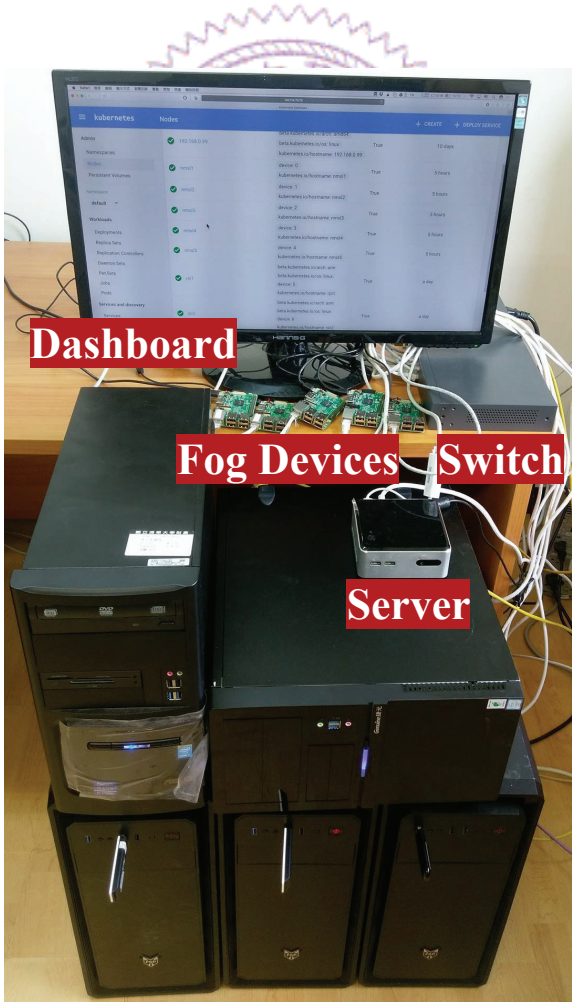


Figure 7.1: Architecture of our experiment setup.

As illustrated in Figure. 7.1, I have implemented a testbed of the proposed fog com-

puting platform. It consists of: (i) an Intel i5 workstation running Ubuntu, Docker, and Kubernetes as the server and (ii) 5 Intel PCs (1.8 GHz 8-core i7 CPUs) and 5 Raspberry Pi 3 model B(1.2 GHz 4-core ARM CPUs) as the fog devices. Raspberry Pis provide several interfaces for us to attach various sensors, such as camera, microphone, and gas sensors. I use HypriotOS [7] as the operating system, which has docker container (version 17.05.0-ce) on it. I implement the software components presented in Fig. 5.1 on the server and fog devices. The server and devices are connected with an Ethernet switch in a private network. In order to emulate WiFi-based IoT networks, I adopt a traffic shaper [20] to throttle the bandwidth at 8 Mbps [27]. To avoid overloading the fog devices, I reserve 20% of resources for operations other than IoT analytics, e.g., OS scheduling and resource monitoring.

I built several Docker images the applications implemented in Sec. 6.2, split the applications into several parts, and put them on multiple devices. To make decisions on splitting our applications, including the Air Quality Monitor and YOLO application, I analyze the code for *possible cutting points*. Take YOLO as an example, YOLO implements neural networks, so the cutting points are located between any two layers, and so is the Sound Classification. For the Air Quality Monitor application, because it support 4 different sensors, the cutting points are between the code for different sensors.

In order to benefit from distributed analytics, I have to continuously process the input data from sensors to fully utilize multiple devices. However, TensorFlow does not support continuously processing. There are three ways to achieve full utilization: (i) multi-threading, (ii) pipeline, and (iii) multiple applications.

In the experiments, if not otherwise specified, I run an application using multiple threads to fully utilize the fog devices. I run each experiment 5 times and present the average results.

7.2 Results

Transfer data to the cloud leads extra network overhead. To emulating the network situation when IoT devices transfer the data from local to the cloud through the Wi-Fi, I use the iperf [8] to measure the network bandwidth from our lab (Taiwan) to Amazon AWS in USA California, and estimate the latency by CloudPing.info [3]. Furthermore, I also calculate the packet loss rate using netstat [21]. After that, I set up a Raspberry Pi as an access point, and install the tool Augmented Traffic Control (ATC) [2] on it. ATC can simulate the network condition easily by config the parameter such as bandwidth, packet loss rate, latency, and etc.

In this experiment, there are two network situations. The bandwidth of the network

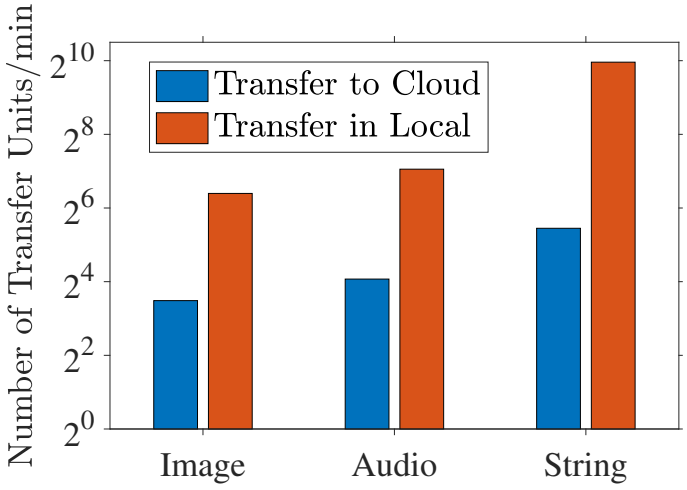


Figure 7.2: Compare the number of transfer units between transfer them to cloud or transfer them in local.

in the local is 18.08Mbits/s. On the other side, the network between local and cloud, the bandwidth is 3.18Mbits/s, latency is 223ms, and the packet loss rate is 0.00279%. I set two PCs which are connected to the Raspberry Pi access point, so I can simulate two network situation I just mentioned. And making one PC keep sending a certain type of data to another one. There are three different types of data: image, audio, and string. The average size of the image is 159KB, audio is 356KB, and JSON struct string is 260Byte. In each iteration of the experiment, I calculate how many transfer units be sent.

Figure 7.2 shows sending data to the devices which are in the local is much efficient than sending data to the remote cloud devices. When the data type is image and audio, the number of transfer data in the local network situation is almost 8 times than in the local to cloud situation. And when the type of the data is the string which is small, the gap between two different network situation is 22 times. In the common situation, cloud side usually has more powerful devices so it can process the data faster, although, this huge network overhead still a critical trade-off which needs to concern carefully.

Docker container leads to low overhead. Virtualization always yields overhead especially on resource-limited devices. We quantify the virtualization overhead in Figure 7.3, which shows that the Docker container virtualization overheads are merely 4.9%, 2.9%, and 1.4% in terms of CPU, RAM, and number of processed images, respectively.

Operators are deployed fast in our platform. Figure 7.4 reports the average deployment time of operators on PCs in each iteration (system models in optimization algorithm will be updated based on the actual resource consumptions, more details are given in our paper [31]). The figure reveals that 37 operators take less than 60 seconds on average to be deployed, which is sufficient for most IoT analytics. IoT analytics that need to have even shorter response time can be *pre-deployed*.

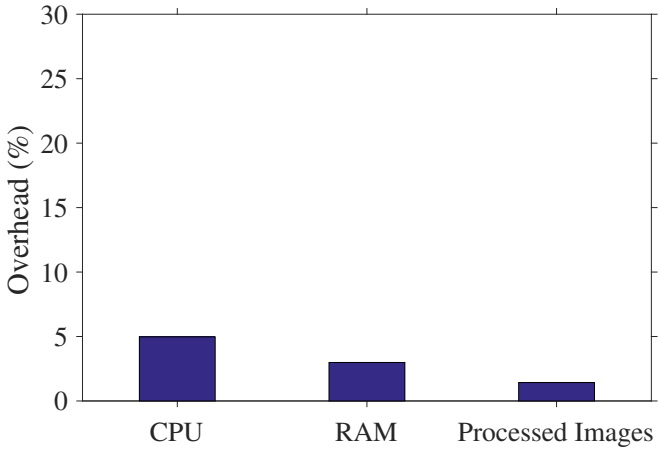


Figure 7.3: Overhead caused by Docker virtualization.

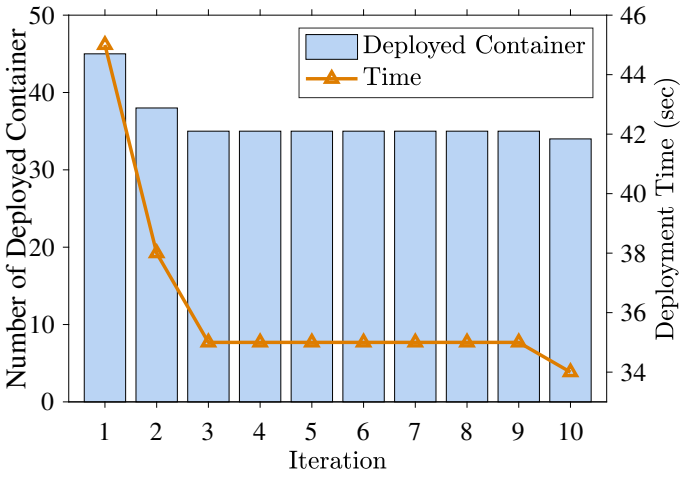


Figure 7.4: Number of deployed container and deployment time in each iteration.

Event-driven in Short Time. I use SCALE box which attached a Passive Infrared (PIR) sensor to responsible for monitoring the motion in the environment and publishes the sensor readings through MQTT [13]. When motion signals become positive, the deployments of the image capture application and object detection application will be triggered. It totally took 32.9 seconds when PIR sensor sensing the motion and then trigger analytic applications and then get the result indicates what object just passes through. And it only tooks 4.8 seconds for event-driven (launches the capture application and object detection application). Comparing to the traditional way that uploads whole surveillance video and checks the video manually, this is more efficient.

TensorFlow achieves low collaboration overhead. We quantify the collaboration overhead, which is handled by TensorFlow automatically in Figure 7.5. To avoid the side-effects due to threading and virtualization, we use a single thread in the experiments. The

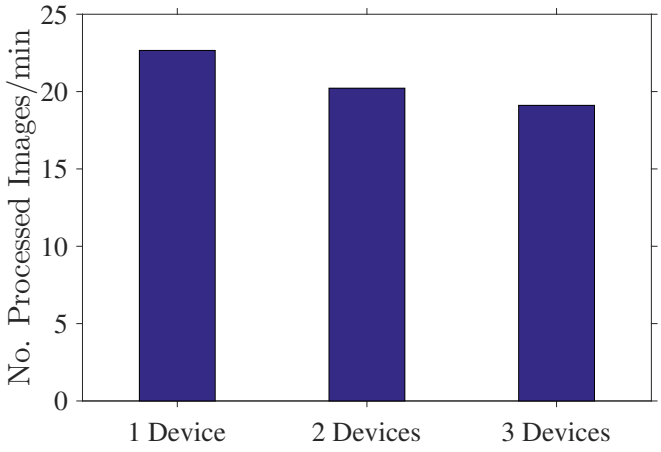


Figure 7.5: Number of processed images per minute without threading and container to quantify collaborating overhead.

results show that adding one more device results to only up to 10% overhead.

Benefits from our distributed analytics. Figure 7.6(a) reports the number of processed data per minute of the Air Quality Monitor application using different number of devices. With this application, distributed analytics does not results in better performance, rather, it leads to worse performance while running in the distributed way. It is because this application's analytics is quite simple: computing moving average. It can be observed in Figure 7.6(b), which shows that the CPU usage always lower then 45%. In summary, such light analytics application can be launched on a single fog device. Running it in the distributed way results in overhead among multiple devices.

For heavy analytics applications, such as our YOLO application, distributed analytics results in large improvements as illustrated in Figure 7.6(c). As shown in the figure, having one more device gives 35.5% and 54.1% improvements. Because this thesis focuses on running state-of-the-art complex analytics applications, we present the results from the YOLO application in the rest of the thesis.

Decisions of cutting an application is critical in distributed analytics. Since we split an application into multiple operators to run it in the distributed way, we next quantify the influence of cutting the application at different points. Figure 7.7(a) shows the number of processed images per minute while YOLO runs on two devices with 8 different cutting points. YOLO implements a 9-layers network, so we make cuts between layers. When the cutting point goes from 1 to 8, more complicated operators are put on the first device. The first device processes images before the second device.

As shown in Figure 7.7(a), cutting points 4 and 5 result in the best performance. It can be explained by Figure 7.7(b), which tells us that cutting an application into smaller

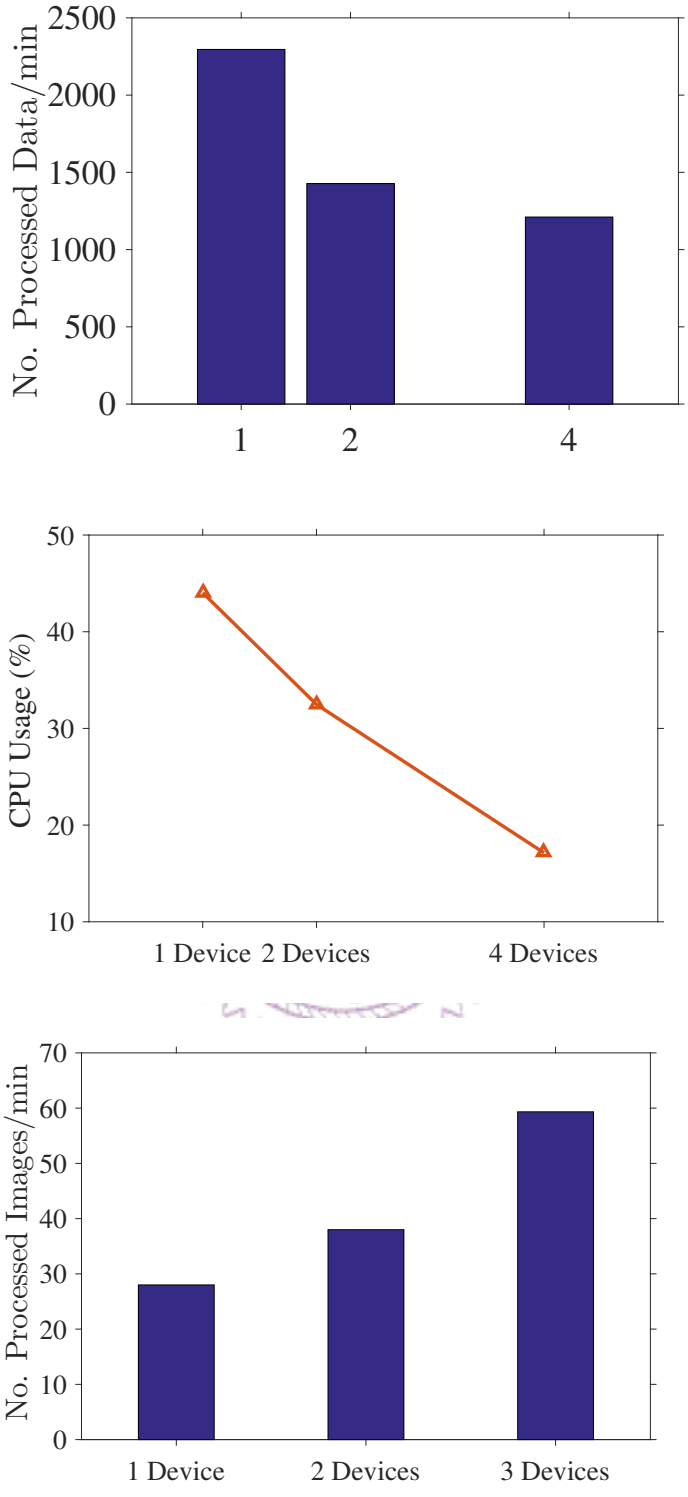


Figure 7.6: Benefits from distributed analytics: (a) the number of processed data of our Air Quality Monitor application, (b) the CPU usage of our Air Quality Monitor application, and (c) the number of processed images of our YOLO application.

operators with equal complexity results in the best performance. Moreover, Figure 7.7(c) reports the network overhead caused by distributed analytics. It shows that if we put

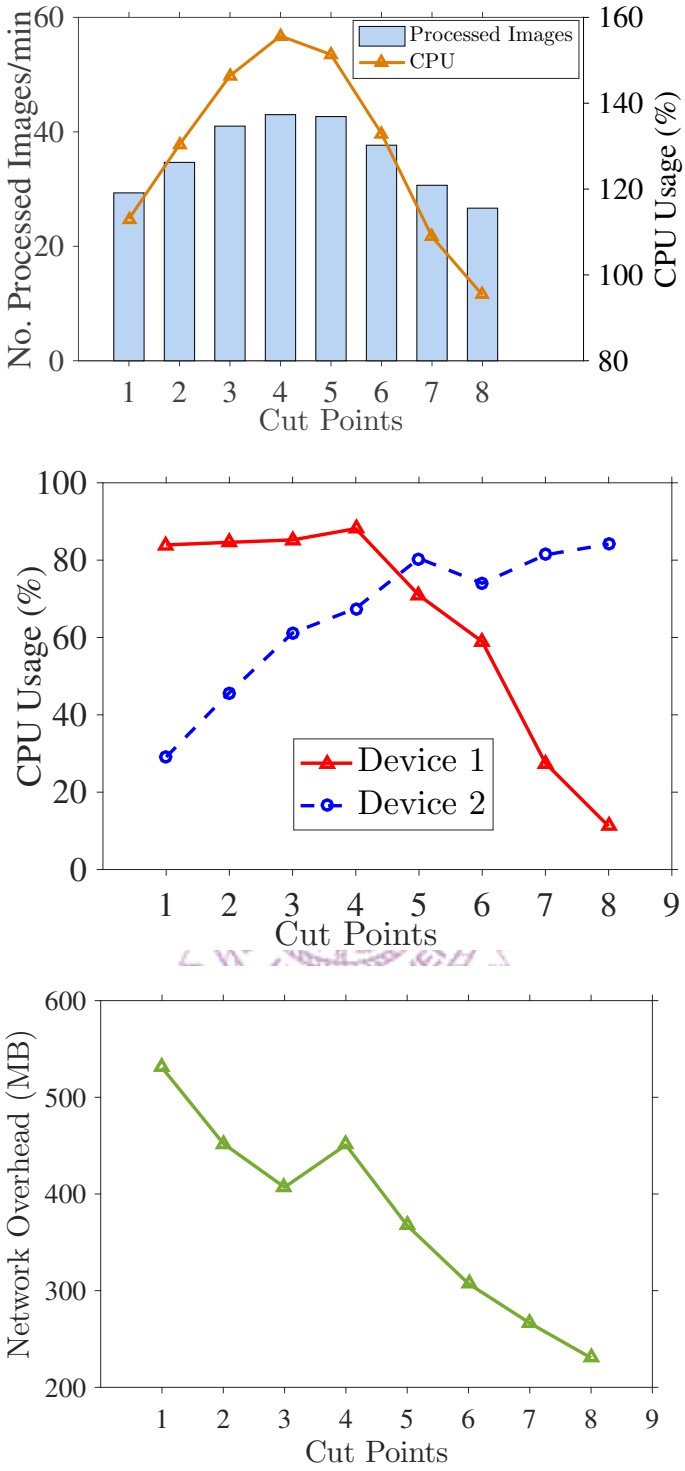


Figure 7.7: Different service quality caused by different cutting points: (a) the number of processed images, (b) the CPU and RAM usages, and (c) the network overhead of our YOLO application.

more loads on the first device, it results in lower network overhead. Hence, when network resources is the bottleneck, we may not prefer equally-loaded splitting decisions. We note that because the YOLO application is CPU-intensive, we do not plot RAM usage, which

only consumes 21.2% (169 MB) on average.

Chapter 8

Related Work

In this chapter, I am going to survey related literature under tow directions: (i) fog computing platform and (ii) edge analytics in the Internet of Things.

8.1 Fog Computing Platform

Several studies [26,41,44] present the concepts/definitions of fog computing. These studies also discuss the challenges, potential applications, and benefits of using fog computing. Building the programming model for fog computing platform is one of the challenging tasks. Hong et al. [34] proposes a hierarchical fog programming model. The first layer of the hierarchical structure is the cloud server and the leaves are end devices. The sensing data collected by the devices can be processed in the nodes which are placed in the middle layers of the hierarchical structure. If it can be processed in a node near to end devices, the latency becomes much lower. However, this work [34] only proposes the high level programming model without considering the practical deployment and implementation issues, so this solution only provides the big picture without practical details.

Designing and implementing fog platforms, which can offload some tasks from end devices to edge servers is studied [36,42]. Cloudlet [36] is proposed before the fog computing concept, but it is somehow similar to the concept of the fog devices. Cloudlet uses a powerful machine, placed near to users for reducing latency, which offers more resources. Users can push virtualized images to the machine to perform their tasks. Unlike my work, the machines running Cloudlets are powerful workstations or small clusters with abundant resources. ParaDrop [42] is a fog device implemented on end-user gateways. ParaDrop framework also has a centralized server to manage the devices. Different from my work, ParaDrop does not consider decomposition of requests and the problem of module deployment.

8.2 Edge Analytics in the Internet of Things

Cisco [26] proposes fog computing, which has been employed for sensor-intensive applications [30] and computational-intensive applications [32]. Many researchers [28, 38, 43] start to leverage the concept of fog computing to address IoT's issues. However, these studies do not consider running applications in a distributed way. For example, our earlier work [32] considers the deployment problem of lighter-weight fog applications, which can be hosted by a single fog device. In contrast, the current paper presents a platform capable to split any fog application into smaller pieces for multiple fog devices.

There are several studies [29, 33, 37] presenting their programming models for distributed fog computing. They also discuss how fog computing benefits IoT regarding its mobility, short response time, and low cost. In particular, Hong et al. [33] propose a PaaS programming model for IoT applications. It supports heterogeneous fog devices and allows applications to dynamically scale in terms of the computing resources. Giang et al. [29] propose a distributed dataflow programming model for IoT applications in fog platforms. Their framework provides an efficient way to develop IoT applications and coordinate distributed resources. Saurez et al. [37] propose a programming model that is implemented using container technology for fog platforms. It provides APIs split applications to communicate with one another. It also includes the algorithms that take the computing resources into account for deployment. Because of the mobility of fog devices, QoS and workload sensitive migrations are supported.

The programming models proposed by these studies are not designed for complex analytics applications in their programming model.

Chapter 9

Conclusion and Future Work

In this thesis, I propose a fog computing platform for distributed analytics. This platform integrates resources from data center to end devices. I leveraged these devices to run IoT and multimedia applications, which take different sensor data, including camera, microphone, and etc. In this thesis, I focused on implementing a platform, which supports complicated analytics, such as Deep Learning to avoid sending a large amount of raw data to powerful data centers for analyzing. I adopt three popular open-source projects, including TensorFlow [18], Docker [4], and Kubernetes [9] to implement my fog computing platform. I conduct extensive measurement studies to quantify the performance. The results show that: (i) Docker container leads to lower than 5% overhead in terms of CPU, RAM, and number of processed images per minute, (ii) operator deployment time, which is less than 20 secs on average in the platform, (iii)running analytics in the distributed way can achieve up to 54.1% performance improvements, and (iv) equal complexity of operators results in the best performance in distributed experiments.

The results in this thesis sheds some lights of distributed analytics on fog devices with limited resources. However, there are some critical issues that have to be considered for my future work. First, the decisions of splitting an application into operators is tricky because different decisions result in different performance and overload. Second, I need to make optimal deployment decisions to serve more IoT applications on the fog computing platform. Third, in order to create an state-of-art IoT ecosystem, I need to let user can operate the fog computing platform more instinctively and the developers can build their own applications running on the platform more flexibly.

Bibliography

[1] Amazon Echo. https://www.amazon.com/Amazon-Echo-Bluetooth-Speaker-with-WiFi-Alexa/dp/B00X4WHP5E.

[2] Augmented Traffic Control. http://facebook.github.io/augmented-traffic-control/.

[3] CloudPing.Info. http://www.cloudping.info.

[4] Docker. https://www.docker.com.

[5] Gartner says 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016. http://www.gartner.com/newsroom/id/3598917.

[6] Gartner says a typical family home could contain more than 500 smart devices by 2022. http://www.gartner.com/newsroom/id/2839717.

[7] HypriotOS. https://blog.hypriot.com.

[8] iperf. https://iperf.fr.

[9] Kubernetes. http://kubernetes.io/.

[10] KVM. http://www.linux-kvm.org/.

[11] librosa. https://github.com/librosa/librosa.

[12] LXC. https://linuxcontainers.org.

[13] MQTT. http://mqtt.org.

[14] Opencv. http://opencv.org.

[15] OpenStack. https://www.openstack.org/.

[16] Saltstack. https://saltstack.com/.

[17] Swarm. https://docs.docker.com/swarm/overview/.

[18] TensorFlow. https://www.tensorflow.org.

[19] Urbansound8k dataset. https://serv.cusp.nyu.edu/projects/urbansounddataset/urbansound8k.html.

[20] Wonder Shaper. http://lartc.org/wondershaper/.

[21] Wonder Shaper. https://linux.die.net/man/8/netstat.

[22] Xen. http://www.xenproject.org/.

[23] Fog computing and the Internet of Things: Extend the cloud to where the things are. `http://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf`, 2015.

[24] K. Ashton. That 'internet of things' thing. *RFID Journal*, pages 97–114, 2009.

[25] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[26] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proc. of ACM Workshop on Mobile Cloud Computing (MCC)*, 2012.

[27] R. Friedman, A. Kogan, and Y. Krivolapov. On power and throughput tradeoffs of WiFi and Bluetooth in smartphones. *IEEE Transactions on Mobile Computing*, 12(2):1363–1375, 2013.

[28] K. Giang, M. Blackstock, R. Lea, and C. Leung. Developing IoT applications in the fog: A distributed dataflow approach. In *Proc of IEEE International Conference on Internet of Things (IOT)*, 2015.

[29] N. Giang, M. Blackstock, R. Lea, and V. Leung. Developing IoT applications in the fog: A distributed dataflow approach. In *Proc. of IoT*, 2015.

[30] H. Hong, J. Chuang, and C. Hsu. Animation rendering on multimedia fog computing platforms. In *Proc. of IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Luxembourg, December 2016.

[31] H. Hong, P. Tsai, A. Cheng, M. Uddin, N. Venkatasubramanian, and C. Hsu. Supporting internet-of-things analytics in a fog computing platform. In *Cloud Computing Technology and Science (CloudCom)*, 2017.

[32] H. Hong, P. Tsai, and C. Hsu. Dynamic module deployment in a fog computing platform. In *Proc. of Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2016.

[33] K. Hong, D. Lillethun, U. Ramachandran, O. B, and B. Koldehofe. Mobile fog: a programming model for large-scale applications on the internet of things. In *Proc. of ACM Workshop on Mobile Cloud Computing (MCC)*, 2013.

[34] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwalder, and B. Koldehofe. Mobile fog: a programming model for large-scale applications on the internet of things. In *Proc. of ACM SIGCOMM workshop on Mobile Cloud Computing (MCC)*, 2013.

[35] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[36] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies. The case for VM-based cloudlets in mobile computing. *IEEE Transactions on Pervasive Computing*, 8(4):14–24, 2009.

[37] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder. Incremental deployment and migration of geo-distributed situation awareness applications in the fog. In *Proc. of ACM International Conference on Distributed and Event-based Systems (DEBS)*, 2016.

[38] S. Shin, S. Seo, S. Eom, J. Jung, and H. Lee. A Pub/Sub-Based fog computing architecture for Internet-of-Vehicles. In *Proc. of IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2016.

[39] M. Y. S. Uddin, A. Nelson, K. Benson, G. Wang, Q. Zhu, Q. Han, N. Alhassoun, P. Chakravarthi, J. Stamatakis, D. Hoffman, L. Darcy, and N. Venkatasubramanian. The SCALE2 multi-network architecture for iot-based resilient communities. In *Proc. of IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–8, May 2016.

[40] I. T. Union. The internet of things—executive summary. *ITU Internet Reports*, 2005.

[41] L. Vaquero and L. Merino. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, 2014.

[42] D. Willis, A. Dasgupta, and S. Banerjee. ParaDrop: a multi-tenant platform for dynamically installed third party services on home gateways. In *Proc. of ACM SIGCOMM Workshop on Distributed Cloud Computing (DCC)*, 2014.

[43] D. Wu, I. Arkhipov, M. Kim, L. Talcott, C. Regan, A. McCann, and V. N. ADDSEN: adaptive data processing and dissemination for drone swarms in urban sensing. *IEEE Transactions on Computers*, 66(2):183–198, 2016.

[44] S. Yi, Z. Hao, and Q. Li. Fog computing: Platform and applications. In *Proc. of IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, 2015.