Distributed Analytics in Fog Computing Platforms Using TensorFlow and Kubernetes

Pei-Hsuan Tsai, Hua-Jun Hong, An-Chieh Cheng, and Cheng-Hsin Hsu
Department of Computer Science, National Tsing Hua University, Hsin-Chu, Taiwan

***Abstract*—Modern Internet-of-Things (IoT) applications produce large amount of data and require powerful analytics approaches, such as 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 this paper, we implement a platform, which integrates resources from data centers (servers) to end devices (IoT devices). We launch distributed analytics applications among the devices without sending everything to the data centers. We analyze challenges to implement such a platform and carefully adopt popular open-source projects to overcome the challenges. We 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 our platform.**

***Index Terms*—IoT, fog computing, virtualization, edge analytics**

I. Introduction

The Internet-of-Things (IoT) is becoming 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 human lifestyle. 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, which is 31% more than that in 2016; and the number will increase to 20.4 billion by 2020 [2].

When the number of IoT devices is getting higher, 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. Sending large sensor data to the data centers 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 features on the IoT devices significantly reduces the amount of network traffic.

Therefore, how to analyze those sensor data and extract 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* [3] , which integrates resources from data centers to end devices to run applications in a distributed way. 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*.

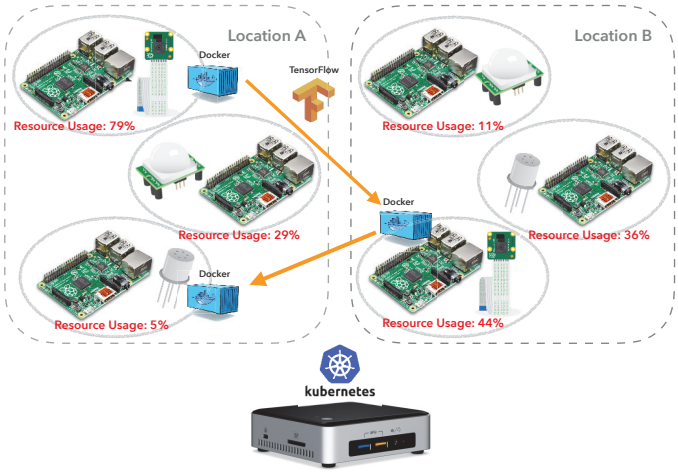


Fig. 1. Overview of our fog computing platform.

To implement such fog computing platforms, 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 IoT 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.

In this paper, we adopt three open-source projects, including TensorFlow [4], Docker [5], and Kubernetes [6] to overcome these challenges and implement a functional fog computing platform. We briefly introduce these three projects in the following:

- **TensorFlow** is a library for numerical computations using data flow graphs. This idea of graphs suits the structure of distributed IoT devices, in which 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

978-1-5386-1101-2/17/$31.00

approaches, which are useful to analyze large amount of sensor data.

- **Docker** is a light-weight virtualization technology compared to traditional virtualization technologies, such as Xen [7] and KVM [8]. 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 our fog computing platform for monitoring fog devices and deploying virtualized applications.

Figure 1 shows our fog computing platform. Kubernetes monitors the information and resource usage, such as CPU usage, memory usage of fog devices (e.g., Raspberry Pi). 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 on multiple fog devices. In this paper, we implement the fog computing platform and conduct extensive experiments to quantify the performance and limitations of the platform. Our 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 for better performance, and (iii) the overhead caused by container is rather low even on resource-limited devices.

In the rest of this paper, we survey the related work in Sec. II. We introduce the overview of our fog computing platform in Sec. III. We give some examples of the IoT applications based on TensorFlow in Sec. IV-B. In Sec. V, we implement the fog computing platform based on container and measure the resource usage of applications in containers. This is followed by our conclusion in Sec. VI.

II. Related Work

Cisco [9] proposes fog computing, which has been employed for sensor-intensive applications [10] and computational-intensive applications [11]. Many researchers [12], [13], [14] 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 [11] 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 [15], [16], [17] presenting their programming models for distributed fog computing. They also discuss how fog computing benefits IoT regards its mobility, short response time, and low cost. In particular, Hong et al. [15] propose a PaaS programming model for IoT applications. It supports heterogeneous fog devices and allows applications to be dynamically scaled in terms of the computing resources. Giang et al. [16] 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. [17] 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.

III. System Overview

In this section, we give an overview on our fog computing platform, including data flow graphs, container technology, applications deployment, and resource monitoring. Fog devices usually don't have enough computing resources to process heavy work, such as DL analytics, on their own. So it is important to have programming models designed for distributed analytics to concurrently leverage resources from multiple fog devices. Because many IoT applications are launched on our fog computing platform, we have to virtualize these applications to isolate applications from one another and reserve heterogeneous resources. Moreover, virtualization also provides benefits of transparent deployment on heterogeneous fog devices. The large number of virtualized applications requires an orchestration tool to manage them. More specifically, while distributing the applications, the strengths and weaknesses of every device must be considered to achieve the best performance. For example, some devices have more CPU resources, so they can be assigned CPU-intensive jobs. On the other hand, devices with more memory spaces have better performance on I/O intensive jobs.

In this paper, we adopt TensorFlow [4] as our programming model, Docker [5] as our virtualization tool, and Kubernetes [6] as our orchestration tool. We describe details of them in the following.

A. TensorFlow

TensorFlow helps us build the data flow graph by splitting an application into multiple smaller *operators*. Operators are the units of deployments. TensorFlow deploys those operators on distributed fog devices and allow 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. Our 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. Such multi-operator applications have several advantages, compared to the single-operator ones. First, multi-operator applications solve the problem that fog devices don't have enough computing resources, since they allow IoT devices to only perform the jobs they are good at. Second, multi-operator applications save the resources and reduce the data transfer amount. For example, if there are two operators on different fog devices needing the same input

data, they can get the data from the same operator, and reuse the operator rather than collecting data separately. Last, pre-processing the raw data 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 data go through the operators, so as to get the results at the edge to reduce the size of data that are transmitted to the

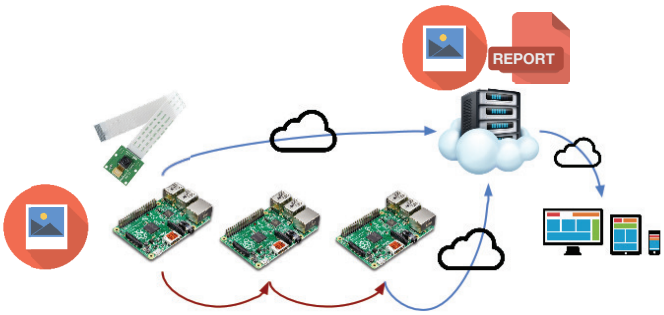


Fig. 2. An example that pre-processes the images before sending them to the cloud data center.

Figure 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, and send the whole image to the data center, more network traffic is generated. Following our fog computing approach, we pre-process the images among the IoT devices and send concise reports to the data center, which reduces the network traffic.

B. Docker

Every operator may require different libraries and 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 quite large for fog devices. Therefore, our platform adopts Docker containers to package the operators with their required libraries into Docker images. Hence, we do not need to install libraries on our fog devices. The operators and libraries are packaged as a container for on-demand deployment.

TABLE I
COMPARISONS BETWEEN CONTAINER AND VIRTUAL MACHINE

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

Similar to traditional virtualization technologies, container technology is the solution to transparently deploy tasks on heterogeneous devices. However, different from traditional virtualization technologies, container is lighter weight and more scalable as summarized in Table I. It is suitable for fog devices that don't have enough computing resources. Besides, the short startup time is also helpful for real-time IoT applications and fast deployments. Furthermore, 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.

C. Kubernetes

For those containers running on heterogeneous devices at different locations, we need a tool to manage and monitor them, such as Kubernetes [6], OpenStack [18], Docker Swarm [19], and SaltStack [20]. We selected Kubernetes because of its popularity. We gain full control over all the operators in every application using Kubernetes. Kubernetes has an extension, named Heapster, which enables container cluster monitoring and resource monitoring. It also provides error recovery. For example, when fog devices run out of batteries or are disconnected from the Internet, Kubernetes can automatically relaunch new containers to keep serving our applications.

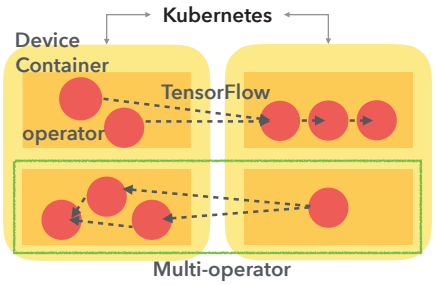


Fig. 3. The architecture of our fog computing platform.

As illustrated in Figure 3, 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 just replace that container, which avoids interrupting other applications.

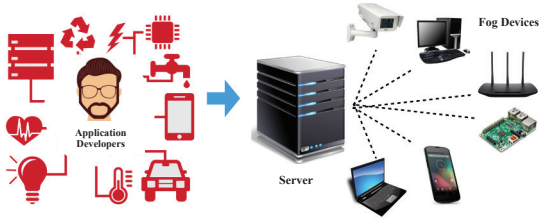


Fig. 4. The ecosystem of our fog computing.

IV. Implementations

In this section, we provide more details on our fog computing platform. This is followed by sample TensorFlow applications.

A. Fog Computing Platform

In the paper, 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 4, this structure makes developers no need to worry about the details of the deployment, so they can focus on the developing of the applications.

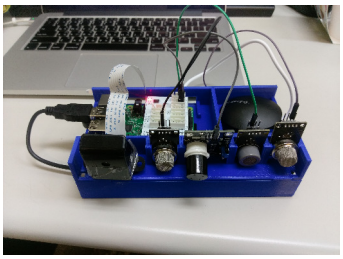


Fig. 5. The SCALE box with various sensors installed.

Fig. 6. A sample result from YOLO.

B. Sample TensorFlow Applications

SCALE [21] (Safe Community and Alerting Network) is a project that aims to use the 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 [22] to a broker. Other 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 is installed the SCALE in a 3D-printed box (illustrated in Figure 5) and attach it to a bike. We collect sensor data along the path where the biker passes. In the rest of this section, we present two applications based on SCALE.

1) Air Pollution Detection: 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, means the air quality is too bad, 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. Our fog computing platform will put those operators on different fog devices, in case of too much burden is incurred on a single device.

2) YOLO: Real-Time Object Detection: You Only Look Once (YOLO) [23] 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 the 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 shows the results we get from an image.

There are two ways to distribute jobs of YOLO. The first one is distributing an image to multiple nodes, which concurrently analyze different grids of the same image. But the YOLO application needs OpenCV library, and this library is too huge to be installed on all nodes. Therefore, in this paper, we use other 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. So every image will be pre-processed by multiple operators among multiple devices as a distributed pipeline. Doing so frees us from filling up the storage space with libraries, and it runs faster if we put the operators on the right devices.

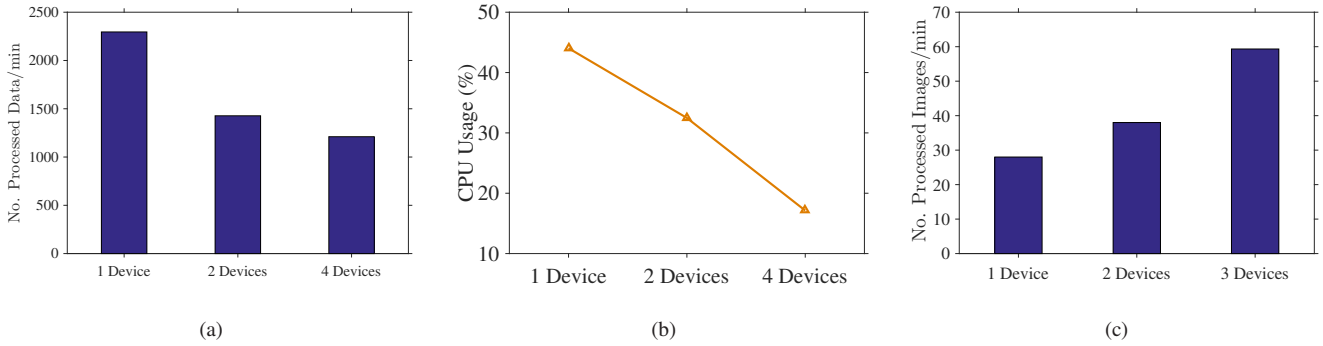
V. Measurement Study

In this section, we conduct real experiments to evaluate our fog computing platform.

A. Experiment Setup

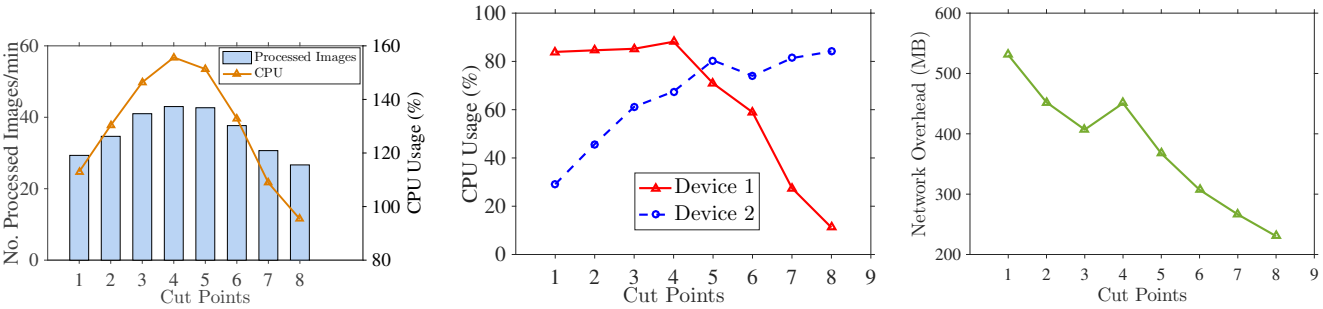
We use Raspberry Pi 3 model B as our fog devices. It has a Cortex-A53 CPU at 1.2GHz and 1GB RAM. Raspberry Pis provide several interfaces for us to attach various sensors, such as camera, microphone, and gas sensors. We use HypriotOS [24] as the operating system, which has docker container (version 17.05.0-ce) on it. Fog devices connect with one another through an Ethernet switch. We built several Docker images for the applications implemented in Sec. IV-B. We split the applications into several parts, and put them on multiple devices. To make decisions on splitting our applications, including the Air Pollution Detection and YOLO application, we 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. For the Air Pollution Detection application, because we support 4 different sensors, the cutting points are between the code for different sensors.

In order to benefit from distributed analytics, we 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, we run an application using multiple threads to fully utilize the



(a) (b) (c)

Fig. 7. Benefits from distributed analytics: (a) the number of processed data of our Air Pollution Detection application, (b) the CPU usage of our Air Pollution Detection application, and (c) the number of processed images of our YOLO application.



(a) (b) (c)

Fig. 8. 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.

fog devices. We run each experiment 5 times and present the average results.

B. Results

Benefits from our distributed analytics. Figure 7(a) reports the number of processed data per minute of the Air Pollution Detection application using different number of devices. With this application, distributed analytics does not results in better performance. In contrast, 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(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(c). As shown in the figure, having one more device gives 35.5% and 54.1% improvements. Because this paper focuses on running state-of-the-art complex analytics applications, we presents the results from the YOLO application in the rest of the paper.

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 8(a) shows the number of processed images per minute when running YOLO 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 8(a), cutting points 4 and 5 result in the best performance. It can be explained by Figure 8(b), which tells us that cutting an application into smaller operators with equal complexity results in the best performance. Moreover, Figure 8(c) reports the network overhead caused by distributed analytics. It shows that if we put 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.

Docker container leads to low overhead. Virtualization always yields overhead especially on resource-limited devices. We quantify the virtualization overhead in Figure 9, 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.

TensorFlow achieves low collaboration overhead. We quantify the collaboration overhead, which is handled by

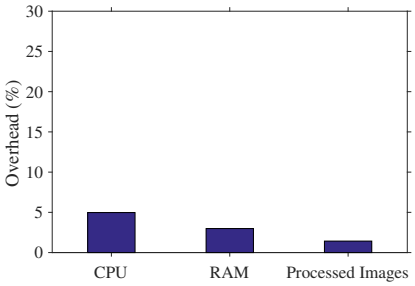


Fig. 9. Overhead caused by Docker virtualization.

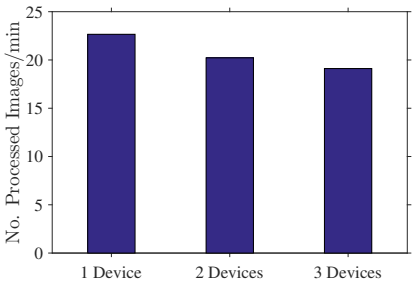


Fig. 10. Number of processed images per minute without threading and container to quantify collaborating overhead.

TensorFlow automatically in Figure 10. To avoid the side-effects due to threading and virtualization, we use a single thread in the experiments. The results show that adding one more device leads to only up to 10% overhead.

VI. Conclusion and Future Work

In this paper, we propose a fog computing platform for distributed analytics. This platform integrates resources from data center to end devices. We leverage these devices to run IoT and multimedia applications, which take different sensor data, including camera, microphone, and etc. In this paper, we focus 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. We adopt three popular open-source projects, including TensorFlow [4], Docker [5], and Kubernetes [6] to implement our fog computing platform. We conduct extensive measurement studies to quantify the performance. The results show that: (i) running analytics in the distributed way can achieve up to 54.1% performance improvements, (ii) equal complexity of operators results in the best performance in distributed experiments, and (iii) Docker container leads to lower than 5% overhead in terms of CPU, RAM, and number of processed images per minute.

The results in this paper sheds some lights of distributed analytics on fog devices with limited resources. However, there are some critical issues that have to be considered as our future work. First, the decisions of splitting an application into operators is tricky because different decisions result in different performance and overload. Second, required resources of each operator will be affected by makes/models of fog devices and different QoS levels, such as sensing frequency. Third, we need to make optimal deployment decisions to serve more IoT applications on our fog computing platform.

Acknowledgements

This work was partially supported by the Ministry of Science and Technology of Taiwan (# 105-2221-E-007-088), and by the Lite-On Technology Corporation.

References

[1] "Amazon Echo," https://www.amazon.com/Amazon-Echo-Bluetooth-Speaker-with-WiFi-Alexa/dp/B00X4WHP5E.
[2] "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.
[3] "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.
[4] "TensorFlow," https://www.tensorflow.org.
[5] "Docker," https://www.docker.com.
[6] "Kubernetes," http://kubernetes.io/.
[7] "Xen," http://www.xenproject.org/.
[8] "KVM," http://www.linux-kvm.org/.
[9] 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.
[10] 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.
[11] 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.
[12] 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*, vol. 66, no. 2, pp. 183–198, 2016.
[13] 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.
[14] 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.
[15] 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.
[16] N. Giang, M. Blackstock, R. Lea, and V. Leung, "Developing IoT applications in the fog: A distributed dataflow approach," in *Proc. of IoT*, 2015.
[17] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwlder, "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.
[18] "OpenStack," https://www.openstack.org/.
[19] "Docker swarm," https://docs.docker.com/swarm/overview/.
[20] "Saltstack," https://saltstack.com/.
[21] 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)*, May 2016, pp. 1–8.
[22] "MQTT," http://mqtt.org.
[23] 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)*, 2016, pp. 779–788.
[24] "HypriotOS," https://blog.hypriot.com.