Optimization of Face Recognition System Based on Azure IoT Edge

Shen Li[1], Fang Liu[1, *], Jiayue Liang[1], Zhenhua Cai[1] and Zhiyao Liang[2]

Abstract: With the rapid development of artificial intelligence, face recognition systems are widely used in daily lives. Face recognition applications often need to process large amounts of image data. Maintaining the accuracy and low latency is critical to face recognition systems. After analyzing the two-tier architecture "client-cloud" face recognition systems, it is found that these systems have high latency and network congestion when massive recognition requirements are needed to be responded, and it is very inconvenient and inefficient to deploy and manage relevant applications on the edge of the network. This paper proposes a flexible and efficient edge computing accelerated architecture. By offloading part of the computing tasks to the edge server closer to the data source, edge computing resources are used for image preprocessing to reduce the number of images to be transmitted, thus reducing the network transmission overhead. Moreover, the application code does not need to be rewritten and can be easily migrated to the edge server. We evaluate our schemes based on the open source Azure IoT Edge, and the experimental results show that the three-tier architecture "Client-Edge-Cloud" face recognition system outperforms the state-of-art face recognition systems in reducing the average response time.

Keywords: Face recognition, edge computing, azure iot edge, computation offloading.

1 Introduction

Face recognition has greatly attracted people's attention because of its application in artificial intelligence for the improvement of people's life. It can be used in many application scenarios, such as public security services, city tracing, photo classification. But the recognition accuracy of the face recognition system has certain requirements for computing resources such as GPU [Dixon, Powers, Song et al. (2015)]. With the limitation of the battery capacity and the computing resources, the mobile devices cannot perform face recognition well [Shi and Dustdar (2016)]. The increasing number of applications on the network has called for a more complex network structure to handle large amounts of data [Cai, Wang, Zheng et al. (2013); Tan, Liu, Wang et al. (2019); Tan, Liu, Xie et al. (2019)]. It requires a more secure network and system architecture for data dissemination and analysis [Liu, Kui and Wang (2004); Liu, Cai, Xu et al. (2015); Teng, Liu, Liu et al. (2019); Liu, Liu, Liu et al. (2019)]. Therefore, most of the face recognition systems are based on the clients-cloud model. With the widespread use of the face

[1] Sun Yat-sen University, Guang Zhou, 510000, China.

[2] The Macau University of Science and Technology, 999078, Macau.

[*] Corresponding Author: Fang Liu. Email: liufang25@mail.sysu.edu.cn.

recognition system in our daily life, more and more photos need to be sent to the cloud for processing, but part of those photos contain no human face and they are not necessary to be sent to the cloud server. If all the photos are sent to the cloud server, massive data transfer between the clients and the cloud will bring high latency and cause network congestion, which will lead to poor user experience. In addition, the security and privacy are also necessary to be concerned in the condition of transmitting large amounts of photos between the client and the cloud [Shi, Cao, Zhang et al. (2016); Shi and Dustdar (2016)].

To decrease the response time of the system, a client-edge-cloud architecture can be used for optimization, and the "edge" is a resource-rich edge device located nearby the client. An edge device is any computing and network resource between the data sources and cloud-based datacenters [Shi and Dustdar (2016)]. However, under the environment of IoT with miscellaneous edge devices, it is very difficult for users to deploy and update the applications on large amounts of edge devices, and the management of these devices is very inconvenient. Moreover, the difference of operating system between the edge devices requires users to develop different versions of the applications which will waste a lot of time. In order to increase the flexibility of the optimization of the face recognition system, a scheme that uses an edge device deployed with Azure IoT Edge as an edge server for optimization will be introduced in this paper. Azure IoT Edge is an open-sourced edge computing system that can be deployed in various edge devices with different operating systems by using the container technology, and it provides an efficient and convenient way for users to deploy and manage their applications on the edge devices. Photos can be sent to the Azure IoT Edge for face detection first when the clients and the edge server are in the same local area network (LAN) with low latency. By filtering out those photos which do not contain human face in Azure IoT Edge, the size of the data needed to be sent to the cloud will be greatly reduced.

The remaining parts of this paper are organized as follows. Section 2 introduces the background of edge computing and several edge computing systems. In Section 3 we will give an introduction of the Azure IoT Edge. Section 4 is the introduction of our optimization scheme for face recognition system based on Azure IoT Edge. And the evaluation will be performed in Section 5. At last, the conclusion is in Section 6.

2 Related work

With the face recognition system gradually being widely used and the rapid emerging of the edge computing, the related work on the optimization of the face recognition system is also ongoing. In Powers et al. [Powers, Alling, Osolinsky et al. (2015)], a mobile-cloudlet-cloud architecture was proposed to perform real-time face recognition by using cloudlet to pre-process data and reduce communication time at the expense of computation. Wang et al. [Wang, Xiong, Pei et al. (2018)] introduces a method to protect the visual privacy by hiding the identity information of the face images. MOCHA [Soyata, Muraleedharan, Funai et al. (2012)] is a mobile-cloudlet-cloud architecture for real-time face recognition that performs task migration from mobile devices to the cloud and dynamically distributes computational load between the cloud and cloud servers. By utilizing the fog nodes to extract the features from raw images, Hu's scheme in Hu et al. [Hu, Ning, Qiu et al. (2017)] reduces the amount of data sent to the cloud servers for face detection and face recognition. And in Hu et al. [Hu, Ning, Qiu et al. (2017)], Hu also proposes a face identification framework based on the fog computing for saving the bandwidth and preserving the security and privacy at the same time.

On the other hand, with the development of the edge computing technology [Shi, Cao, Zhang et al. (2016); Zhao, Liu and Cai et al. (2018)], more and more companies and open source communities have proposed their system and platform for edge computing to extend today's cloud computing infrastructure. The MIST Lab at Wayne State University proposed a programming model for edge computing, Firework [Zhang, Zhang, Zhang et al. (2016)]. Apache Edgent, which was known as Apache Quarks previously, is an Apache Incubator project at present. It is an open source programming model and lightweight embedded streaming analytics runtime that can be used in some small devices on the edge. In 2017, Linux Foundation published EdgeX Foundry which is a standardized interoperability framework developed for industrial IOT edge computing, whose sweet spots are edge nodes such as gateways, hubs, routers, etc. Azure IoT Edge [Gremban (2018)] is edge computing framework published by Microsoft and aims to give a more intelligent solution at the edge of the network. All these systems provide solutions to the various problems that arise with IoT applications. In order to deal with the problem in the face recognition system, Azure IoT Edge will be selected for the optimization. In Section 3, an introduction about Azure IoT Edge and the reason for using it to optimize the face recognition system will be introduced.

3 Open-source azure iot edge

3.1 Introduction of azure iot edge

Developing or migrating an application to amounts of edge devices will meet many challenges. Due to the variety of hardware devices, it is very difficult for users to develop a general version of the application. Moreover, in the complicated environment of the network edge, the applications deployed on the edge devices often require frequent modifications according to the actual needs on business. It is quite inconvenient for users to update the applications deployed in a large number of edge devices. When an error appears at the edge of the network, it is hard to find out in which edge device.

To solve the problems in developing applications at the edge of network, Microsoft proposed Azure IoT Edge. Azure IoT Edge is an open-source system for users to deploy applications more efficient on the edge device. Users can rewrite the source code of Azure IoT Edge according to specific needs in various hardware facilities. And Azure IoT Edge use the container technology to run application on edge device which solves the difficulty of developing applications on various hardware devices. The deployment and management on massive edge devices in large scale can also be achieved by using the Azure IoT Hub which is a cloud service for IoT devices management provided by Microsoft Azure. In Azure IoT Hub, users also can monitor the running situation of the edge devices and accurately locate the device with the runtime error. By utilizing Azure IoT Edge on the edge devices, users can easily extend or migrate our application which is originally deployed on the cloud to the edge of network. For dealing with the problem that it is too difficult to manage a large amount of IoT devices, IoT Edge provides a convenient way for users to manage and deploy the multiple devices in Azure portal which is a control interface for users to manage their IoT devices.

3.2 Architecture of azure iot edge

Azure IoT Edge has two main components: Edge Module, Edge Runtime [Gremban (2018)]. Its architecture is shown in Fig. 1.

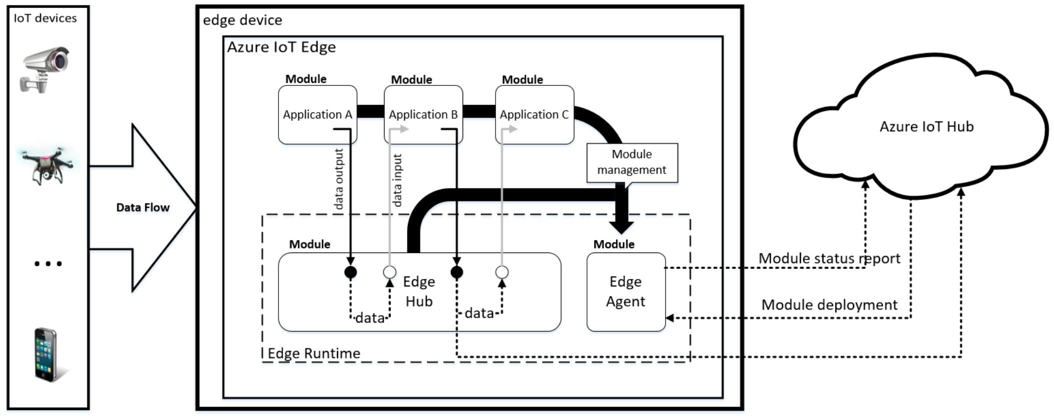

Figure 1: The architecture of Azure IoT Edge

3.2.1 IoT edge module

A service on the edge device usually consists of several edge modules, and each module is responsible for different functions. In order to easily manage the modules on the edge device, Azure IoT Edge use Moby which is an open framework created by Docker as its management system for containers. Every edge module (we can call it a module instance) running on the edge device can be seen as a container created by an edge module image. A module instance is equivalent to a docker container, and a module image is equivalent to a docker image.

Each module instance is related to a module twin which is a JSON document that records the module's metadata, configuration, and the status. When a module was deployed on the device, a module twin will be built on the IoT Hub, and it is used for synchronizing the state and configuration of the module from the IoT Hub.

Azure IoT Edge allows users to deploy some complicated computation modules by using Azure services, and users also can create their own modules. If users want to create their own modules, firstly, they can use VSCode to create a new edge module, write their own code on it and create it as an image file, then push it to the Docker Hub or Azure Container Registry. Secondly, users can deploy their modules on one or several edge devices by using VSCode or IoT Cloud Interface, and the Edge Runtime will automatically start the modules as module instances in the edge device.

The features of the edge modules are as follows: 1) Migrating the application which was originally running on the Cloud to the edge device can be easily achieved by building an edge module with the original code of the application. 2) Deploying the service on multiple devices becomes more efficient because an edge module can be quickly deployed in different edge devices. 3) Each module runs as a single container, which effectively reduces the dependencies between modules. A module's running error does not affect the other modules.

3.2.2 IoT edge runtime

The IoT Edge Runtime is mainly responsible for installing and updating the workloads, keeping the security of Azure IoT Edge and reporting the situation of the module to the cloud. Moreover, it also monitors the communication inside Azure IoT Edge.

Edge Runtime is mainly divided into two system modules, Edge Hub and Edge Agent. The Edge Hub manages the communication inside the IoT Edge. The Edge Agent manages the working situation of modules. They both run as an Edge Module on the edge device like the others. About Edge Runtime we can see Fig. 1.

Edge Hub: The Edge Hub runs as a local proxy for IoT Hub on the edge and manages the connection between devices and the Cloud Center. Each device who wants to connect need Edge hub to forward authentication requests to IoT Hub on the cloud. Meanwhile, Edge Hub is responsible for the communication between IoT devices and Edge Modules, and the communication between Edge Modules within the device. The data from a module or an IoT device will be output to the Edge Hub as a message queue with a unique name defined by developers. And the other modules who want to get data can register an input with a unique name too. And then the developers need to declare the routes about the data from which output queue to which input queue by using their unique name. By using the routes declared in the deployment manifest, the data from a module or an IoT device can also be output to the IoT Hub on the cloud. Edge Hub will control the transmission of messages according to the routes. We can learn from Fig. 1 how the Edge Hub monitors the communication inside IoT Edge.

Edge Agent: The Edge Agent is mainly responsible for the management of the modules in the device. It needs to instantiate the other modules, keep them running, and feed back the status of the modules to the IoT Hub. After the edge device is started, Edge Agent will also be started by the Edge security manager and get its module twin from IoT Hub and inspect the deployment manifest. Deployment manifest is a JSON document, it tells the Edge Agent which modules need to be started, which image to use to instantiate the module instance, and some advanced operations required for instantiation. After starting all the modules, Edge Agent also need to keep them running, monitor the runtime status about each module and report the status to the IoT Hub. When the modules have the runtime error, Edge Agent need to restart the module according to the restart policy declared in the deployment manifest.

4 Design and implementation

4.1 Architecture overview

As for the face recognition system, we use *ageitgey/face_recognition* which is a powerful, simple and easy-to-use face recognition open source project. It is based on the *dlib* and the model has an accuracy of 99.38% tested on the *Labeled Faces in the Wild* [King (2009); Learned-Miller, Huang, Roychowdhury et al. (2016)]. It provides APIs for face detection and face recognition. The process of the face recognition in *ageitgey/face_recognition* is composed of Face Detection (FD) and Face Recognition (FR). FD will detect the human face in an image and return the information of the face locations if the image contains the human face. Then FR will extract the face features according to the face locations from the image and compare with the face features already known whom it belongs to.

Baseline scheme: The baseline scheme is based on the client-cloud model. The *ageitgey/face_recognition* is used as a face recognition system running on the cloud server. When a client sends images to the cloud server for face recognition, firstly the images will be processed with the FD. If the image does not contain the human face, it will be dropped out and the cloud server will return the result to the device. The rest of the images containing human face will be sent to the FR. The face features extracted from the images will be compared with the face features stored in the cloud server, and the cloud server will feed back the name of the people recognized from the images. In the baseline scheme, FD and FR are all processed on the cloud server.

Optimization scheme: In order to greatly reduce the data transmitted in the long network link between the client and the cloud server, the FD can be migrated from the cloud server to the edge server for filtering out those images which don't contain the human face. Clients need to send all the images to the edge server first, and FD will be performed on all photos. Those images without human face will be dropped out by edge server, and the others will be sent to the cloud for further face recognition. Because the clients and the edge servers are mostly in the same LAN, the distance between the clients to the edge server is much shorter than that from the clients to the cloud, sending all the images to the edge server is much faster. By filtering out those useless images, the size of the images needed to be sent to the cloud will be greatly reduced, and the response time of the face recognition will also be significantly decreased.

4.2 Design detail

The introduction of the design detail and implementation of the optimization scheme are as shown in Fig. 2.

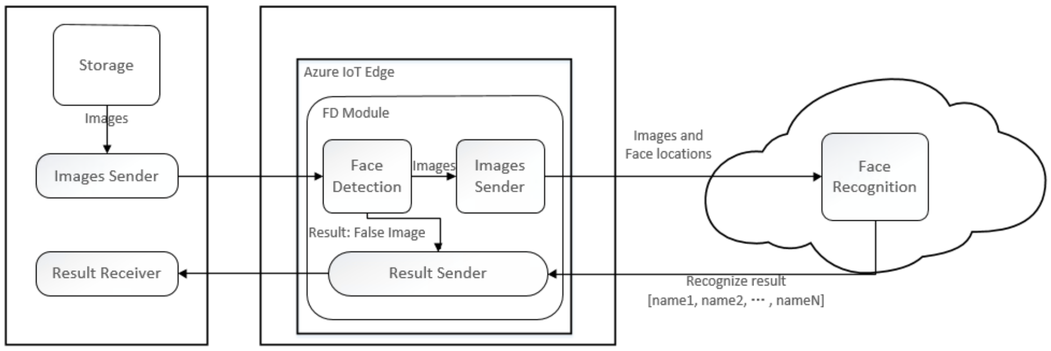

Figure 2: The architecture of the Optimization Scheme

Clients: In this experiment, two scenarios using face recognition system are simulated. For the first one, in the smart-phone, the photo album usually needs to identify the people in the photos for photo classification. And in the second one, the photos collected by the city surveillance cameras will be analyzed by the face recognition system for finding the fugitives. In these two scenarios, the clients all need to send the photos to the face recognition system for recognizing the target people in the photos. Four Android-phone are used as clients to post the photos to the face recognition system, and the system will feed back the names of the target people found in the images to the clients.

Edge server: An edge module is created with FD, and the Azure IoT Hub is used to deploy the module on an edge server which contains Azure IoT Edge already. The FD module will keep running and waiting for the images sent by the clients. FD module is composed of three parts which are Face Detector, Result Sender and Images Sender. Detector will detect the images receiving from the clients and drop out the useless images, and the detection result of the useless images will be delivered to the Result Sender. The rest of the images and the information of the face locations extracted from them will be delivered to the Images Sender, and Images Sender sends these images and their face locations information to the cloud server for further processing. All result received by the Result Sender will be sent back to the client.

Cloud server: Cloud server stores a set of data of face features which are extracted from multiple people. In the baseline scheme, the images receiving from the clients will be processed with FD and FR, and the extracted face features will be compared with the already known face features. And the result of the FD and FR will be fed back to the clients directly. In optimization scheme, because the useless images have been filtered out and the face locations are delivered with the images together, it is only necessary to process these images with FR, and the result of FR will be returned to the Result sender in edge server.

5 Performance evaluation

5.1 Experiment platform and dataset

To evaluate the performance of the optimization face recognition system, the implementation of the Baseline Scheme and optimization Scheme will be performed in the experiment. The experiment platforms are as shown in Tab. 1. Four Android-based smartphones are used as the clients to send images for simulating multi-user usage scenarios. In the reality, the edge server is close to the data source, and mostly the clients and the edge server are on the same LAN which means that distance between the clients and the edge server is very short. The clients and the edge server used in the experiments are in the same LAN of our laboratory, therefore the transmission delay between the clients and the edge server is so small that can be out of consideration.

Table 1: Platforms

	Number	Type
Clients	4	Android-based smartphones 8x Qualcomm Kryo 260 CPU 2.2 GHz 8 GB RAM
Edge server	1	Ubuntu 16.04 Intel Core i5-8400, 2.80 GHz, 8 GB RAM (Azure IoT Edge)
Cloud server	1	Alibaba Cloud, Ubuntu 16.04 Intel Xeon (Skylake) Platinum 8163, 4 vCPU, 16 GB RAM, 2.50 GHz

As for the experimental data set, two kinds of the images are collected, one kind of them is the images containing buildings and the streets except for the human face, another is the images contain a clear human face. We call these two kinds of images as *NCF* (do Not Contain human Face) and *CF* (Contain human Face). All of the images are collected from the Baidu Image, and the average size of each image is 120 KB. We divide these images into four types of image sets with 1400 images including the *NCF* and *CF*. For evaluating the performance of the two schemes when the image set has different ratio of NCF, each type of image set has the different ratio of the NCF and the total size of each type of image set is about 144 MB. The ratio of NCF in each type of image set is 20%, 40%, 60%, and 80% respectively as we can learn in Tab. 2. In the experiments, considering the situation in reality that the number of images sent by the different clients is different, each type of the image set will be divided into four clients with the different number of images as shown in Tab. 2. We will proceed with four experiments for the baseline Scheme and optimization Scheme respectively under different ratio of NCF.

Table 2: Image dataset

The ratio of NCF	Number of images in client1	Number of images in client2	Number of images in client3	Number of images in client4
20%	200	500	400	300
40%	300	200	500	400
60%	400	300	200	500
80%	500	400	300	200

In order to evaluate the performance of two different scheme of the face recognize system in the different network environment, we will proceed the experiments mentioned above under three different network bandwidth which is 200 KB/s, 400 KB/s and 600 KB/s respectively.

5.2 Result of the experiments

The experiments are proceeded under the environment of 600 KB/s network bandwidth first, and the result is shown in Fig. 3.

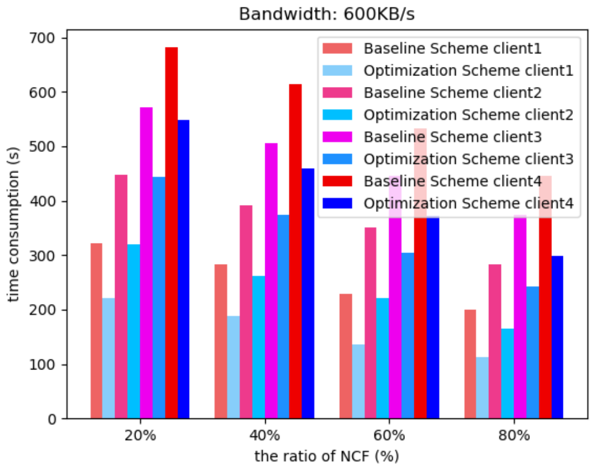

Figure 3: The time consumption of face recognition in each client of two scheme

In Fig. 3, the y-axis is the time consumption of face recognition to all images in each client under different ratio of NCF. And we can see in Fig. 3, the time consumption of the optimization Scheme has a significant reduction which is about 25% compared to the baseline Scheme, which means the optimization Scheme has faster processing speed. With the increasement of the ratio of NCF, the number of CF gradually decreases which means less images need to be processed with PR. Therefore, the total time consumption of processing all images in each client also decreases as we can see in Fig. 3.

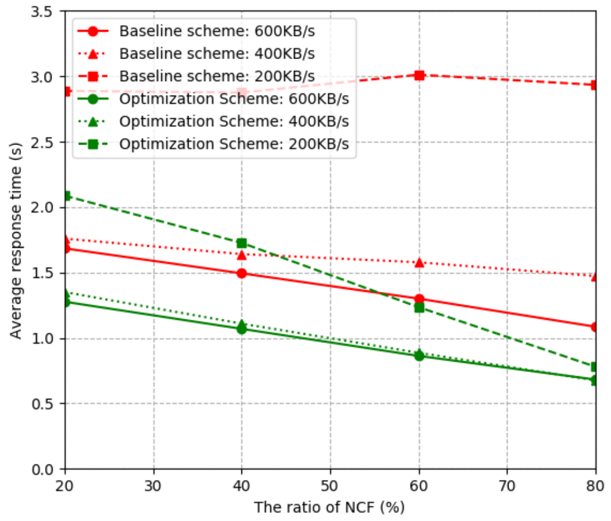

Figure 4: Average response time

The most obvious and straightforward performance metric is the average response time of each image that refers to the average delay in the clients sending images to the edge server or cloud server and receiving the recognition result. From the Fig. 4 we can see the average response time of two schemes in the different conditions of the ratio of NCF and network bandwidth. It is quite obvious that the average response time of the optimization scheme is much shorter than the baseline scheme. When the network bandwidth is 200 KB/s, it means that the network has congestion, and the baseline scheme has longer average response time than the optimization scheme. But on the contrary, the optimization scheme can still perform the low latency as usual, and this will offer a better user experience even though the network condition is terrible.

Table 3: The time consumption of FD and FR

	Total time consumption (s) (600 KB/s)				Time consumption of FD (s)				Time consumption of FR (s)			
	20%	40%	60%	80%	20%	40%	60%	80%	20%	40%	60%	80%
Baseline scheme	2021	1792	1506	1302	1177	1087	997	913	582	433	274	122
Optimization scheme	1531	1283	1034	817	710	662	613	564	528	391	249	108

In addition, the time consumption of FD and FR is also detected. As shown in Tab. 3, we can find that the time consumption of the FD and FR takes a huge part in the total response time, which means accelerating the processing speed of FD and FR plays a vital role in the optimization of the face recognition system. In Tab. 3, the time consumption of FD in optimization scheme is less than the baseline scheme. In the optimization scheme, the processing of FD has been migrated from the cloud server to the edge server. Due to the differences in hardware resources between the edge server and cloud server, the FD processing in the edge server is faster than it processing in the cloud server.

Mostly, the cloud server needs to provide various hardware resources to support miscellaneous services, and the resources that each service can get are limited. Different from the cloud server, the services on the edge server are more specific which means each service can get more specific resources for accelerating the processing speed.

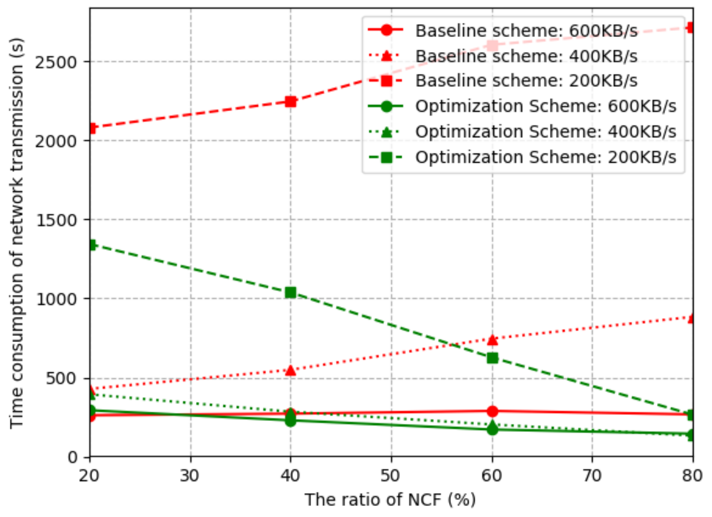

Figure 5: The time consumption of network transmission

According to the two solid lines on the bottom of Fig. 5, and comparing with the two schemes in the case of 600 KB/s network bandwidth, we can find that the time consumption of network transmission can get a better reduction when the ratio of NCF is bigger than 20%. In the case of the other two kinds of network situation, the optimization scheme's time consumption of network transmission is less than that of baseline scheme. It is worth to note that, when the network bandwidth is 200 KB/s, the baseline scheme takes a lot of time for network transmission because of the network congestion. On the contrary, the time consumption of optimization scheme is much smaller than that of baseline scheme. In conclusion, the optimization scheme is better than the baseline scheme.

6 Conclusions

This paper introduces the wide application of face recognition system (FRS) and its bottleneck. Nowadays, most of the FRS is based on the client-cloud model, and the massive data transmitted between the client and the cloud will bring a lot of problems, such as high latency, network congestion, security and privacy. The development of edge computing provides an opportunity to solve these problems in client-cloud FRS. This paper proposes an optimization design of a face recognition system based on edge computing, and a prototype of the system based on Azure IoT Edge, an open source platform, is implemented. Based on the actual data set, the experimental results show that our design is superior to the state-of-art client-cloud face recognition system under different network conditions. In face recognition processing, the time consumption of the proposed scheme is about 25% less than that of baseline scheme. The utilization of edge resources makes a great contribution to improve the processing speed of face recognition and greatly reduces the cost of network transmission.

Acknowledgement: This work is supported by the National Key Research and Development Program of China under Grant (No. 2016YFB1000302) and the National Natural Science Foundation of China under Grant (No. 61832020).

References

Cai, Z.; Wang, Z.; Zheng, K.; Cao, J. (2013): A distributed TCAM coprocessor architecture for integrated longest prefix matching, policy filtering, and content filtering. *IEEE Transactions on Computers*, vol. 62, no. 3, pp. 417-427.

Dixon, W.; Powers, N.; Song, Y.; Soyata, T. (2015): Theoretical foundation and GPU implementation of face recognition. *Enabling Real-Time Mobile Cloud Computing through Emerging Technologies*.

Wang, X.; Xiong, C.; Pei, Q.; Qu, Y. (2018): Expression preserved face privacy protection based on multi-mode discriminant analysis. *Computers, Materials & Continua*, vol. 57, no. 1, pp. 107-121.

Gremban, K. (2018): What is Azure IoT Edge.

https://docs.microsoft.com/en-us/azure/iot-edge/about-iot-edge.

Hu, P.; Ning, H.; Qiu, T.; Song, H.; Wang, Y. et al. (2017): Security and privacy preservation scheme of face identification and resolution framework using fog computing in internet of things. *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1143-1155.

Hu, P.; Ning, H.; Qiu, T.; Zhang, Y.; Luo, X. (2017): Fog computing-based face identification and resolution scheme in internet of things. *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 1910-1920.

King, D. E. (2009). Dlib-ml: a machine learning toolkit. *Journal of Machine Learning Research*, vol. 10, no. 3, pp. 1755-1758.

Learned-Miller, E.; Huang, G. B.; Roychowdhury, A.; Li, H.; Hua, G. (2016). Labeled faces in the wild: a survey. *Advances in Face Detection and Facial Image Analysis.*

Liu, Y.; Liu, A.; Liu, X.; Huang, X. (2019): A statistical approach to participant selection in location-based social networks for offline event marketing. *Information Sciences*, vol. 480, pp. 90-108.

Liu, F.; Kui, D.; Wang, Z. (2004): Improving security architecture development based on multiple criteria decision making. *Content Computing*, pp. 214-218.

Liu, S.; Cai, Z.; Xu, H.; Xu, M. (2015): Towards security-aware virtual network embedding. *Computer Networks*, vol. 91, pp. 151-163.

Powers, N.; Alling, A.; Osolinsky, K.; Soyata, T.; Zhu, M. et al. (2015): The cloudlet accelerator: Bringing mobile-cloud face fecognition into real-time. *IEEE Globecom Workshops*.

Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. (2016): Edge computing: vision and challenges. *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637-646.

Shi, W.; Dustdar, S. (2016): The promise of edge computing. *Computer*, vol. 49, no. 5, pp. 78-81.

Soyata, T.; Muraleedharan, R.; Funai, C.; Kwon, M.; Heinzelman, W. (2012): Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. *IEEE Symposium on Computers and Communications*.

Tan, J.; Liu, W.; Wang, T.; Xiong, N.; Song, H. et al. (2019): An adaptive collection scheme-based matrix completion for data gathering in energy-harvesting wireless sensor networks. *IEEE Access*, vol. 7, pp. 6703-6723.

Tan, J.; Liu, W.; Xie, M.; Song, H.; Liu, A. et al. (2019): A low redundancy data collection scheme to maximize lifetime using matrix completion technique. *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, pp. 5.

Teng, H.; Liu, Y.; Liu, A.; Xiong, N. N.; Cai, Z. et al. (2019): A novel code data dissemination scheme for internet of things through mobile vehicle of smart cities. *Future Generation Computer Systems*, vol. 94, pp. 351-367.

Zhang, Q.; Zhang, X.; Zhang, Q.; Shi, W.; Zhong, H. (2016): Firework: big data sharing and processing in collaborative edge environment. *Fourth IEEE Workshop on Hot Topics in Web Systems and Technologies*.

Zhao, Z.; Liu, F.; Cai, Z.; Xiao, N. (2018): Edge computing: platforms, applications and challenges. *Journal of Computer Research & Development*, vol. 55, no. 2, pp. 327-337.