Poster: Container-Based Architecture for Optimal Face-Recognition Tasks in Edge Computing

Nadim Tellez
Systems Engineering Department
Universidad del Norte
Barranquilla, Colombia
barreran@uninorte.edu.co

Miguel Jimeno
Systems Engineering Department
Universidad del Norte
Barranquilla, Colombia
majimeno@uninorte.edu.co

Augusto Salazar
Systems Engineering Department
Universidad del Norte
Barranquilla, Colombia
augustosalazar@uninorte.edu.co

Elias Nino-Ruiz
Systems Engineering Department
Universidad del Norte
Barranquilla, Colombia
enino@uninorte.edu.co

***Abstract*— Edge Computing has been proposed as an architecture for offering services to mobile devices or sensors. Modern multimedia applications such as face recognition, object and pose identification, and mobile augmented reality require cloud computing resources close to the mobile and sensors devices. Docker and containers have been proposed as a platform for edge computing and as a tool for efficient service handoff across edge computing servers. This paper presents a container-based cloudlet environment to improve resource usage for the architecture using optimization algorithms.**

Keywords: component, formatting, style, styling, insert

I. INTRODUCTION

Edge computing offers services to mobile devices or sensors, for which cloud services' response times do not bring the desired quality of service. This architecture offers nodes at the edge to run code instead of running it on the resource-restrained devices. The impact of cloud data centers to multimedia applications has been measured before [1]. Multimedia applications that have gained attention such as face recognition, object and pose identification, and mobile augmented reality, are benefited when cloud computing resources are close to the mobile and sensors devices. This requirement gave space for the emergence of technologies to solve this, where the main difference is in the purpose for which such technologies are built. One of the most popular is edge computing, which is driven by the necessity of building edge data centers closer to the mobile and sensors devices based on virtual machines, back in 2009 [2]. Authors have differentiated edge computing [3] from fog computing [4]. Fog computing on the other side has been proposed mainly for decentralizing IoT infrastructure by creating multiple layers of nodes between the edge devices (usually sensors) and the cloud. Cloudlet architectures can be compared in several metrics, which include cost, scalability, mobility support, freedom of service node to move to other cloudlets, and computation duration. Docker and containers have been proposed as a platform for Edge Computing [5] and as a tool for efficient service handoff across edge computing servers [6]. Experiments of using containers showed that their proposal reduced the total duration of service handoff time by 80%. Our work uses containers as a platform for cloudlets with the ability to select the best place to execute the tasks using a task allocation algorithm. We present a container-based cloudlet environment to improve resource usage using an optimization algorithm in face recognition tasks based on OpenFace.

II. THE CONTAINER-BASED CLOUDLET DESIGN

This cloudlet architecture allows image classification services to be executed either on the cloudlet or in the cloud depending on the resources available. The cloudlet uses a task allocation algorithm to select the best distribution of resource assignment according to the tasks and resources available at the time of the execution. The purpose is to enable lightweight cloudlets based on Docker which target resource-constrained devices as their deployment platform. Given this constraint, the necessity of a task allocation algorithm becomes clearer. Figure 1 shows a container-based architecture where cloudlet devices and cloud devices can resolve tasks according to the type of each one. For example, classification and training tasks are resolved directly in the cloud-based OpenFace. However, the container-based cloudlet layer will resolve the face recognition tasks. It is clear that tasks should be allocated and resolved by each layer depending on the weight or complexity of it. This kind of approach is a static and non-optimized task allocation architecture. We propose this same architecture but not only executing face recognition tasks in the cloudlet layer but also executing classification and training tasks.

III. ARCHITECTURE IMPLEMENTATION

We implemented the architecture to test an optimization algorithm we proposed in a previous paper [7]. The figure shows three main layers of the architecture. The first layer is composed of the end user devices such as smart-phones and laptops. These devices are the ones who demand the service provided by the superior layers.

The next component is the optimization algorithm which runs as a part of the cloudlet layer. Every task that is demanded by the users is analyzed by this algorithm to make the allocation decisions according to the optimization. When the optimization process completes, it executes and distributes the tasks according to the results of the algorithm. The Gurobi library [8] runs as the core of the LIP algorithm. The Cloudlet layer is composed of 4 Fog nodes; we implemented them as container devices running in Docker. Each of them is running a public Docker image called "bamos/openface" with an add-on of the Flask Library which allows each fog node to publish a web service [9]. Each fog node runs a Python script which

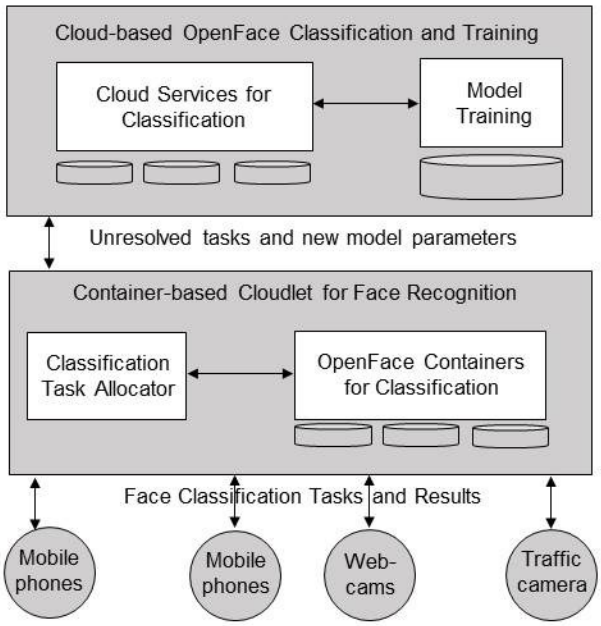

Fig. 1. Container-based cloudlet architecture for face recognition

launches a web service to execute the OpenFace task according to the user requirement.

The third layer is representing a cloud computing scheme. In this layer, two cloud nodes are running as Docker OpenFace containers. This layer was implemented inside an Ubuntu virtual machine loaded in Microsoft Azure platform. Each of the cloud nodes is almost the same as the ones running in the cloudlet layer. However, the main difference between every node in the architecture is the container capacity. In the case of the cloudlet containers, the CPU was limited to 2 processors and the RAM to 500 MBytes. In the cloud containers, the limitation was set to 2 processors and 4 GBytes of RAM. Disk and bandwidth were also different for each type of node. The algorithm needs the cost of every task measured as resources used to make the correct task assignment for each node. Working with the OpenFace library, we could be able to extract and measure three main jobs. Table I summarizes and describes the resource consumption of each type of task.

The tasks that this implementation will be using are the following. Training: these tasks expect as input images to build and train the system to build a "classifier." The classifier file has all the information related to the face recognition that allows the algorithm to recognize, identify, and classify other faces that were not used to train the system. Specifically for this project, this task is executing a training set of 30 images of two people. Classification: is the process of identifying and establishing a level of confidence in an image concerning a face recognition process. For this experiment, this task is classifying two images to get the level of confidence according to the classifier generated by the training process. Compare: it allows the user to compare 2 or more images. There is no need to train or recognize a person but to compare how close one face to another face is. This example task is comparing four different images of two different people. The main idea is to measure how close is one face of another and giving a confidence rate to measure the comparison.

TABLE I. TYPES OF TASKS IN OPENFACE

Task Type	*RAM (MB)*	*CPU Used*	*Cores Used*	*Disk Used (MB)*	*Time (sec.)*
Classify	178.2	65.3%	1.31	1	6.15
Train	358.2	91.9%	1.84	5.3	14.93
Compare	199.3	76.8%	1.54	1.5	6.308

TABLE II. TASK ASSIGNMENT RESULTS

Task	*Task Type*	*Assigned Node*
1	Train	Fog 1
2	Classify	Fog 2
3	Classify	Fog 3
4	Compare	Fog 4
5	Classify	Cloud 1
6	Classify	Cloud 2

IV. EXPERIMENTS AND RESULTS

For the implementation, we built two examples to demonstrate how the optimization algorithm and the architecture are working. We used four Fog nodes and two cloud nodes. Each node is emulated by a Docker image running Ubuntu with the OpenFace library. We prepared a list of tasks requirements per experiment. The clients send the tasks requirements, and a centralized node receives them. This centralized node executes the optimization algorithm. Then, this centralized application is going to distribute and allocate each task to its correspondent node according to the algorithm results. This training process consists in running the LIP optimization processes 1000 times while the value of is being modified each cycle randomly. The outputs of this operation are the nodes where each task is going to be assigned and the total cost of the optimized response. As a result of the experiment, the architecture and the optimization algorithm allocated the tasks in the order shown in Table II where the first four tasks were allocated in the cloudlet layer, and the rests were allocated in the cloud layer.

V. FUTURE WORK

As a continuation of this work, we plan to create a new version of the optimization algorithm proposed in [7] where the parameters will adapt to the OpenFace library requirements. We also plan to run a new set of experiments with more tasks as input to have more detailed results. The input for the architecture will also be modified to receive a stream of tasks and not only a list of predefined tasks as these experiments used. We consider that this work highlights the potential of using container-based cloudlet nodes with optimization algorithms to better allocate OpenFace tasks.

REFERENCES

[1] K. Ha *et al.*, "The Impact of Mobile Multimedia Applications on Data Center Consolidation," in *2013 IEEE International Conference on Cloud Engineering (IC2E)*, 2013, pp. 166–176.

[2] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct. 2009.

[3] M. Satyanarayanan, "The Emergence of Edge Computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017.

[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, New York, NY, USA, 2012, pp. 13–16.

[5] B. I. Ismail *et al.*, "Evaluation of Docker as Edge computing platform," in *2015 IEEE Conference on Open Systems (ICOS)*, 2015, pp. 130–135.

[6] L. Ma, S. Yi, and Q. Li, "Efficient Service Handoff Across Edge Servers via Docker Container Migration," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, New York, NY, USA, 2017, p. 11:1–11:13.

[7] N. Tellez, M. Jimeno, A. Salazar, and E. Nino-Ruiz, "A Tabu Search Method for Load Balancing in Fog Computing," *Int. J. Artif. Intell.*, vol. 16, no. 2, Oct. 2018.

[8] "Gurobi Optimization - The State-of-the-Art Mathematical Programming Solver." [Online]. Available: http://www.gurobi.com/. [Accessed: 26-Jul-2018].

[9] "Welcome | Flask (A Python Microframework)." [Online]. Available: http://flask.pocoo.org/. [Accessed: 26-Jul-2018].