LAVEA: Latency-aware Video Analytics on Edge Computing Platform

Shanhe Yi, Zijiang Hao, Qingyang Zhang[†‡], Quan Zhang[†], Weisong Shi[†], Qun Li
College of William and Mary, [†]Wayne State University, [‡]Anhui University, China
{syi,hebo,liqun}@cs.wm.edu, [†]{qyzhang, quan.zhang, weisong}@wayne.edu

***Abstract*—**
We present LAVEA, a system built for edge computing, which offloads computation tasks between clients and edge nodes, collaborates nearby edge nodes, to provide low-latency video analytics at places closer to the users. We have utilized an edge-first design to minimize the response time, and compared various task placement schemes tailed for inter-edge collaboration. Our results reveal that the client-edge configuration has task speedup against local or client-cloud configurations.

I. Introduction

Edge computing is proposed to overcome inherent problems of cloud computing and power the Internet of Things (IoT) [1]–[5]. Among many edge applications, we focus on video edge analytics. The ability to provide low latency video analytics is critical for applications in the fields of public safety, counter-terrorism, self-driving cars, VR/AR, etc [6]. For example, in "Amber Alert", our system can automate and speedup the searching of objects of interest by vehicle recognition, plate recognition and face recognition utilizing web cameras deployed at many places. Simply uploading or redirecting video feeds to cloud cannot meet the requirement of latency-sensitive applications. Because computer vision algorithms such as object tracking, object detection, object recognition, face and optical character recognition (OCR) are either computation intensive or bandwidth hungry. In addressing such problems, Mobile cloud computing (MCC) utilizes both the mobile and cloud for computation. An appropriate partition of tasks that makes trade-off between local and remote execution can speed up the computation and preserve energy at the same time. However, there are still concerns of cloud about the limited bandwidth, the unpredictable latency, and the abrupt service outage. Existing work has exploited adding intermediate server (cloudlet) between mobile client and the cloud. Cloudlet is an early implementation of the cloud-like edge computing platform with virtual machine (VM) techniques. We employed a different design on top of lightweight OS-level virtualization which is low-cost, modular, easy-to-deploy/manage, and scalable.

In this paper, we are considering a mobile-edge-cloud environment and we put most of our effort into the mobile-edge and inter-edge side design. To demonstrate the effectiveness of our edge computing platform, we have built the **L**atency-**A**ware **V**ideo **E**dge **A**nalytics (LAVEA) system. We divide the response time minimization problem into two sub-problems. First, we formulated computation offloading problem as a mathematical optimization to choose offloading tasks and allocate bandwidth among clients. Second, we enable inter-edge collaboration by leveraging nearby edge nodes to reduce the overall task completion time. We investigated task placement schemes and the findings provided us insights that lead to an efficient predication-based task placement scheme.

II. System Design

We present our system design in Figure 1. Our design goals are: 1) **Latency**. The ability to provide low latency services is recognized as one of the essential requirements of edge computing system design. 2) **Flexibility**. Edge computing system should be able to flexibility utilize the 3) **Edge-first**. By edge-first, we mean that the edge computing platform is the first choice of our computation offloading target.

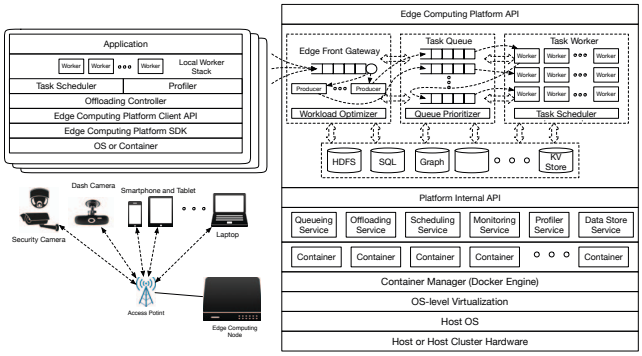

Fig. 1: The architecture of edge computing platform

In LAVEA, the edge computing node attached to the same access point or base station as clients is called the *edge-front*.

Edge-front offloading. We consider N clients and only one edge server. Each client $i, i \in [1, N]$ processes tasks belong to a certain job, e.g. recognizing plates, and select tasks for offloading to the edge. Without loss of generality, we start with a directed acyclic graph (DAG) $G = (V, E)$ as the task graph. Each vertex $v \in V$ is the computation of a task (c_v), while each edge $e = (u, v), u, v \in V, e \in E$ represents the intermediate data size (d_{uv}). The remote response time includes the transmission delay of sending data to the edge server and the execution time. We use an indicator $I_{v,i} \in \{0, 1\}$ to indicate the task v of client i running locally or remotely. The total local execution time for client i is $T_i^{local} = \sum_{v \in V} I_{v,i} c_v / p_i$ where p_i is the processor speed of client i. Similarly, we use $\overline{T}_i^{local} = \sum_{v \in V} (1 - I_{v,i}) c_v / p_i$ to represent the execution time

1063-6927/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDCS.2017.182

of running the offloaded tasks locally. For network, when there is an offloading decision, the client need to upload the intermediate data (outputs of previous task, application status variables, configurations, etc) to the edge server in order to continue the computation. The data transmission delay is modeled as $T_i^{net} = \sum_{(u,v)\in E}(I_{u,i} - I_{v,i})d_{uv}/r_i$ where r_i is the bandwidth assigned for this client. For each client, the remote execution time is $T_i^{remote} = \sum_{v\in V}(1 - I_{v,i})(c_v/p_0)$ where p_0 is the processor speed of the edge server.

The offloading task selection problem can be formulated as

$$\min_{\mathbf{I}_i, r_i} \quad \sum_{i=1}^{N}(T_i^{local} + T_i^{net} + T_i^{remote}) \tag{1}$$

where the task selection is represented by the indicator matrix **I**. The optimization problem is subject to constraints: 1) The total bandwidth s.t. $\sum_{i=1}^{N} r_i \leq R$ 2) We restrict the data flow to avoid ping-pong effect: s.t. $I_{v,i} \leq I_{u,i}, \quad \forall e(u,v) \in E, \forall i \in [1,N]$ 3) Unlike mobile cloud offloading, we consider the resource contention or schedule delay at the edge side by adding a end-to-end delay constraint. s.t. $T_i^{local} - (T_i^{net} + T_i^{remote}) > \tau, \quad \forall i \in [1,N]$ where τ can be tuned to avoid selecting borderline tasks that if offloaded will get no gain due to the resource contention or schedule delay at the edge.

Inter-edge Collaboration. The intuitive task placement scheme for inter-edge collaboration is to transfer tasks to the candidate edge node which has the least number of queued tasks upon the time of query, stated as shortest queue length first (SQLF). However, this scheme neglects the network latency and has scalability issue when the number of candidates scales. We have designed a novel predication-based shortest scheduling latency first (SSLF) scheme for its ability to estimate the scheduling latency efficiently. To measure response time, edge-front appends a no-op task to the task queue of each candidate edge node. When the special task is executed, the edge-front shall receive the response message and maintain a series of response times for each candidate. Since the candidate's workload may vary from time to time, the most recent response time cannot serve as a good predictor of the response time. The edge-front estimates the response time for each candidate by regression analysis of history response time series. In this way, edge-front node can offload tasks to the edge node with the estimated least response time. Once the edge-front node start to place task to a certain candidate, the estimation will be updated using piggybacking of the redirected tasks, which lowers the overhead of measuring.

III. Evaluation

We have built a testbed consisting of four edge computing nodes. We make use of two types of Raspberry Pi (different models with different network interfaces) nodes as clients, We employed three datasets: 1) Caltech Vision Group 2001 dataset, 2) a video contains rear license plates in various resolutions, 3) a small dataset in the OpenALPR project.

To understand the benefit of offloading tasks, we design an experiment on wired and wireless clients. The result of the first case is straightforward: the clients upload all the data and run all the tasks remotely in edge offloading or in cloud offloading, as shown in Fig. 2. The result of wireless client node offloading tasks to the edge or the cloud is in Fig. 3. Overall, our results show that the client-edge configuration has a speedup ranging from 1.3x to 4x (1.2x to 1.7x) against running in local (client-cloud).

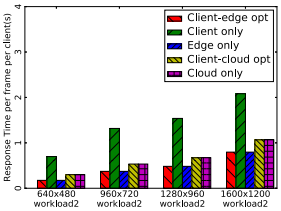

Fig. 2: The comparison of task selection impacts on edge offloading and cloud offloading for wired clients (RPi2).

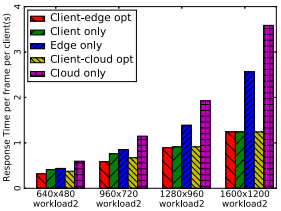

Fig. 3: The comparison of task selection impacts on edge offloading and cloud offloading for 2.4 Ghz wireless clients (RPi3).

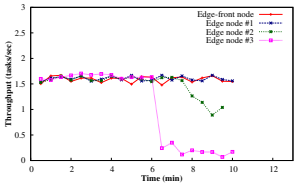

Fig. 4: Performance of SQLF.

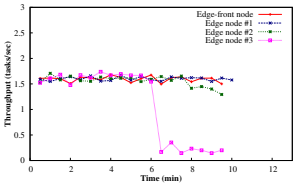

Fig. 5: Performance of SSLF.

In evaluating inter-edge collaboration, Fig. 4 and Fig. 5 illustrate the throughput result of SQLF scheme and SSLF scheme respectively. In the setup, edge node #1 has the lowest transmission overhead but the heaviest workload among the three edge nodes, while edge node #3 has the lightest workload but the highest transmission overhead. Edge node #2 has modest transmission overhead and modest workload. In SQLF, the edge-front node transmits tasks to less-saturated edge nodes, efficiently reducing the workload on the edge-front node. However, the edge-front node intends to transmit many tasks to edge node, which has the lowest bandwidth and the longest RTT to the edge-front node. SSLF scheme considers both the transmission time of the task and the waiting time in the queue on the target edge node, and therefore achieves the better performance.

References

[1] M. Satyanarayanan, "Pervasive computing: Vision and challenges," *IEEE Personal communications*, vol. 8, no. 4, pp. 10–17, 2001.

[2] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data, Mobidata '15*. ACM, 2015, pp. 37–42.

[3] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *Hot Topics in Web Systems and Technologies (HotWeb), 2015 Third IEEE Workshop on*. IEEE, 2015, pp. 73–78.

[4] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[5] Z. Hao, E. Novak, S. Yi, and Q. Li, "Challenges and software architecture for fog computing," *Internet Computing*, 2017.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct 2016.