Enabling Real-Time AI Edge Video Analytics

Vassilis Tsakanikas
SuITE Research Group
Department of Computer Science
London South Bank University
London, UK
tsakaniv@lsbu.ac.uk

Tasos Dagiuklas
SuITE Research Group
Department of Computer Science
London South Bank University
London, UK
tdagiuklas@lsbu.ac.uk

***Abstract*—This paper introduces a novel distributed AI model for managing in real-time, edge based intelligent analytics, such as the ones required for smart video surveillance. The novelty relies on distributing the applications in several decomposed functions which are linked together, creating virtual chain functions, where both computational and communication limitations are considered. Both theoretical analysis and simulation analysis in a real-case scenario have shown that the proposed model can enable real-time surveillance analytics on a low-cost edge network. Finally, a caching mechanism is proposed and evaluated, reducing further the operational costs of the edge network.**

***Index Terms*—edge computing, AI applications, Virtual Function Chaining, caching, cost optimization**

I. Introduction

Artificial Intelligence (AI), as expressed by the latest developments of Machine Learning (ML) and Deep Learning (DL), has produced numerous models which are mature enough to reach the market massively during the next few years. The main reasons for this, mainly involves the improvements on data-capturing devices, the re-engineering of several ML algorithms and the release of ML and DL toolkits, like PyTorch and TensorFlow [1]. Video analytics is an umbrella term for describing applications like object tracking, pedestrian detection, face recognition, behavioral analytics etc. The common business - technology model for deploying AI surveillance services nowadays is Cloud Computing [2], where the captured video streams are uploaded to a centralized data center. This imposes a round-trip time to the throughput of the service which, in many cases reduces significantly the Quality of Service (QoS). This leads the service providers to either reduce the processing frames per second (fps) or lower the resolution of the captured videos, nullifying the advances of new video sensors, like UHD and HDR sensors. Edge Computing has been proposed as a computing paradigm according to which the data are processed 'near' the generating data devices and comprises many low-capacity devices [3]. While this paradigm addresses the large round-trip times of Cloud Computing, the QoS is now limited due to the capacity of the edge devices. Not only academia, but lately industry has placed focus on Edge Computing, by providing software (e.g. Google Lite TensorFlow©) and hardware (e.g. NVIDIA Jetson AGX Xavier© and AWS DeepLens©) solutions are suitable for edge processing. This paper proposes a novel distributing framework which explores the Virtual Function Chaining (VFC) concept inspired from the Software-Defined Networks and enables the real-time inference for surveillance applications at the Edge. In this model, an AI smart video analytics service can be decomposed to a set of Virtual Functions (VFs), which can be deployed on different edge devices. Using these VFs, a VFC is created which process the streaming data in a distributed fashion.

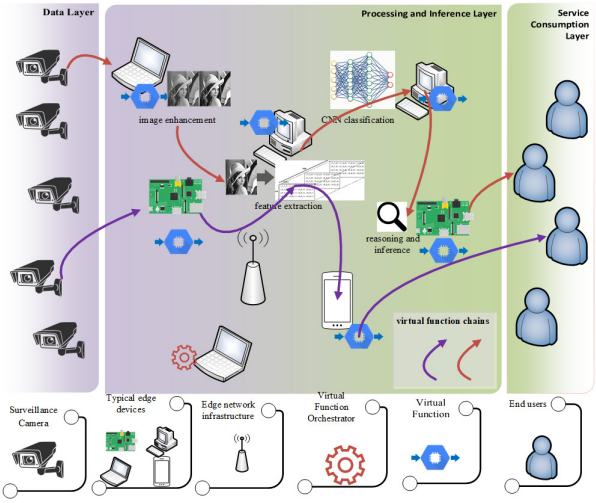

Fig. 1. Conceptual architecture of the proposed system.

The contributions of this work mainly include: (i) A model for designing the optimum setup for a VFC, in terms of VF instances, VF placement on the edge devices. Each VF may appears in the VFC at several instances and interconnected $VFCs$. (ii) A system architecture which seamlessly integrates $VFCs$. The proposed architecture's services are mainly hosted on the Virtual Function Orchestrator (VFO). (iii) An optimization framework for effectively deploying AI video surveillance services on the Edge. (iv) A prototype of the described architecture, which is used to evaluate the described models and provide a proof of concept, in terms of effectiveness and feasibility, on enabling $VFCs$ as a model for real-time AI surveillance applications. (v) A caching mechanism, which demonstrates the scalability of the proposed model when multiple services are required.

While the proposed model is inspired by the Service Function Chaining (SFC) concept, it alters and extends several

of its features, in order to meet the requirements of video analytic services. First, it introduces a load-balancing mechanism connected with the desired QoS, which monitors the output of the service and rearranges the VFC automatically. Additionally, it extends the SFC model by allowing one VF instance to be part in several $VFCs$ (e.g. face detection, gender identification, etc.).

Several research studies have been focused on enabling edge computing to support demanding latency sensitive applications. Author in [4] has proposed a scheme for handling mass video data coming from city surveillance services on heterogeneous digital devices. Zhou *et al.* [5] have described a model for offloading cloud by utilizing an edge meshed network. Li *et al.* [6] have proposed a general virtualization architecture, based on VMs, mainly focusing on its networking aspects. Chen *et al.* [7] have described an architecture which explores fog computing as a processing infrastructure for supporting dynamic urban surveillance streams. Dautov *et al.* [8] have performed a comparison study among cloud, fog and edge computing for supporting intelligent surveillance applications. The authors in [9] provides a survey of the applications that can be supported from edge computing. Author in [10] provides a holistic vision about surveillance applications on edge / fog computing paradigms, where the basic concepts, challenges and opportunities are discussed. Finally, Chen *et al.* [11] have proposed a distributed deep learning model for video surveillance systems, while Puthal *et al.* [12] have presented a novel approach on load balancing of distributed edge servers. Our model has compared with the current state-of-the art literature and does not require any special virtualization (e.g. Virtual Machines) [6] or distribution [13] (e.g. Apache Spark©) middle-ware in order to perform the real-time calculation of AI analytics, offloading the edge devices from the substantial overhead these approaches require. Additionally, surveillance applications are decomposed in virtual functions that are deployed in nodes with the available resources. Such functions are scaled up based on demands. The rest of this paper is organized as follows. Section II describes the VFC model while Section III includes the system implementation details. In Section IV, the results from the experiments are discussed and the conclusions and future work are drawn in Section V.

II. Model formulation

Aiming to enable edge as a real time inference mechanism for AI video analytics services, a generalization framework is proposed, according to which a video analytic service is decomposed to a set of VFs, creating a VFC. The proposed model aims to facilitate the efficient offloading of surveillance cloud services to a cooperative distributed edge environment. The basic principles of the proposed model (Figure 1) are the following: (i) Each surveillance service is decomposed to set of processes. Each process implements certain tasks, like image enhancement, edge detection, etc. (ii) Each process instantiates a VF and is deployed as an edge node. Each VF comprises three parallel threads, the Input Queue, the Output Queue and the Running agent. (iii) A surveillance service is realized by a VFC, similar to the service function chaining proposed by the IETF WG [14]. A VFC must include at least one instance of each VF. The main concept of the model proposed by [14] includes network services, like firewall and packet filtering. (iv)The VFO manages the VFs allocation to the physical devices and established the communication channels among them. Additionally, VFO monitors the performance criteria of the service (e.g. processed frames/sec, total cost, etc.) and performs actions in order to meet them. Each service is described by a VFC, and in general, a single VF can have multiple instances within the edge environment. When a user subscribes to a service (e.g. object detection, etc.), VFO instantiates the VFC by implementing the following tasks: (i) Calculates the required number of instances for each VF, in order to meet the service's QoS criteria, (ii) allocates the VFs to edge devices and (iii) establishes communication channels between the edge devices. A VF instance can be part of several service chains. Table I summarizes the formulations of the main entities of the proposed modeling framework.

TABLE I
Basic entities of the proposed model.

Entity	Formulation	Description
Surveillance service	$\overrightarrow{S} = [fps, \{VF_j\}]$	fps is the processed frames / sec the service requires (QoS)
		$\{VF_j\}$ is the set of processes comprise the service
Virtual Function	$\overrightarrow{VF} = [cpu_{load}, outdata]$	cpu_{load} is the required CPU instructions per frame
		$oudata$ is amount of data virtual function produces after processing the input data
Edge Node	$\overrightarrow{K} = [m, c(l), r(l)]$	m is the CPU instructions / sec the device can execute
		$c(l)$ is the cost function of the device, when performing l CPU instructions. Cost is a general term which includes battery life, maintenance cost, etc.
		$r(l)$ is the required time to process l CPU instructions
Link	$\overrightarrow{W_a b} = [bw]$	bw is the communication bandwidth among edge nodes a and b

A. Problems definition

VFO node needs to assign the VFC to the edge environment. In order to achieve this, the following general assignment constrained problem needs to be solved:

Problem 1*: Determine the number of VF instances and assign each instance to an edge device, such that the video analytics are generated while maintaining the required fps, and the total network cost be minimum.*

Problem 1 can be formulated as:

$$min \left\{ \sum_{i=1}^{n} \sum_{j=1}^{y} x_{i,j} c_{i,j} \right\} (1) \quad min \left\{ y = \sum_{j=1}^{m} instances_j \right\} (2)$$

$$\sum_{j=1}^{y} x_{i,j} = 1 (3) \qquad \sum_{i=1}^{n} x_{ij} \leq 1 (4)$$

$$time_{computational} + time_{communication} \leq \frac{1}{fps} (5), with$$

$$time_{computational} = \sum_{i=1}^{n} \sum_{j=1}^{y} x_{i,j} t_{i,j}$$

$$time_{communication} = \sum_{i=1}^{n} \sum_{i'=1}^{n} \sum_{j=1}^{y-1} x_{i,j} x_{i',j+1} \frac{output_j}{W_{i,i'}}$$

, where n is the number of edge devices, m is the number of the VFs, $x_{ij} = \begin{cases} 1 & \text{if } VF_j \text{ is assigned on node } n_i \\ 0 & \text{otherwise} \end{cases}$, t_{ij} is the required time for device D_i to execute VF_j and process the data produced from a single frame and $instances_j$ is the number of the required instances for VF_j. Finally, $W_{i,i'}$ is the bandwidth of the communication link between nodes i and i', which host two adjacent VFs, j and $j+1$. This formulation describes a model which aims to minimize the total cost of the service (eq. 1) while meeting the QoS constrains (eq. 5), with the minimum number of VF instances (eq. 2), requiring that each VF instance must by assigned to exactly one edge device (eq. 3) and each edge device can undertake no more than one VF instance (eq. 4). This is a NP-Hard problem [15], which requires a substantial computational time to be solved. In order to acquire a feasible solution within an acceptable timeframe, we decouple *Problem 1* to two sub-problems: (A) VF instances sub-problem and (B) the assignment (placement) sub-problem.

Sub-problem A aims to identify the minimum number of instances for each VF. As discussed in the previous section, one instance from each VF must be deployed on the VFC, in order to support the service. Let $GVF = [VF_1, VF_2, ..., VF_n]$ be the set of the first instances of each VF. Each one of these VFs will be deployed to a different edge device. At this point of the assigning workflow, the allocation cost is not considered. Yet, we seek if there is a feasible solution of the placement, such that the QoS constrain is met. This results to the following relaxation.

$$min \left\{ \sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij} t_{ij} + \sum_{i=1}^{n} \sum_{i'=1}^{n} \sum_{j=1}^{y-1} x_{i,j} x_{i',j+1} \frac{output_j}{W_{i,i'}} \right\} \quad (6)$$

$$\sum_{j=1}^{m} x_{ij} = 1 \quad (7) \qquad \sum_{i=1}^{n} x_{ij} \leq 1 \quad (8)$$

Regarding t_{ij}, it can be calculated using the $r_d()$ function of the edge device d. Thus $t_{ij} = r_i(N_j)$, where N_j is the CPU instructions required by VF_j to complete its task. (Eq. 7) reflects the fact that each VF from the GVF set must be appointed only to one node and (eq. 8) that each edge node can not undertake more than one VF. *Sub-problem A* can be solved in a polynomial time by modeling it as a Min Cost Flow problem, which is a widely studied problem [16]. The utilized solver is a typical Hungarian algorithm. This process results to an allocation of the GVF with the minimum required time that the network can support. Let t^* be the resulting time. If $t^* \leq \frac{1}{fps_{serv}}$, then the edge network can support the service without having to replicate a subset of the VFs. In this case, we can re-formulate the assignment problem as a constrained mixed integer problem, with the following formulation

Sub-problem B:

$$min \left\{ \sum_{i=1}^{n} \sum_{j=1}^{m} x_{i,j} c_{i,j} \right\} (9) \quad \sum_{j=1}^{m} x_{i,j} = 1 (10)$$

$$\sum_{i=1}^{n} x_{i,j} \leq 1 (11) \quad \sum_{i=1}^{n} \sum_{j=1}^{m} x_{i,j} t_{i,j} +$$

$$\sum_{i=1}^{n} \sum_{i'=1}^{n} \sum_{j=1}^{m-1} x_{i,j} x_{i',j+1} \frac{output_j}{W_{i,i'}} \leq \frac{1}{fps_{serv}} (12)$$

The objective function (eq. 9) of this formulation aims to minimize the total cost of the VFC deployment to edge network, while fulfilling the QoS constrains of the service (eq. 12) and assigning all VFs to a device (eq. 10) while limiting the number of deployed VFs to a device (eq. 11). This problem, can be solved by utilizing Constrained Programming [17] in a polynomial time.

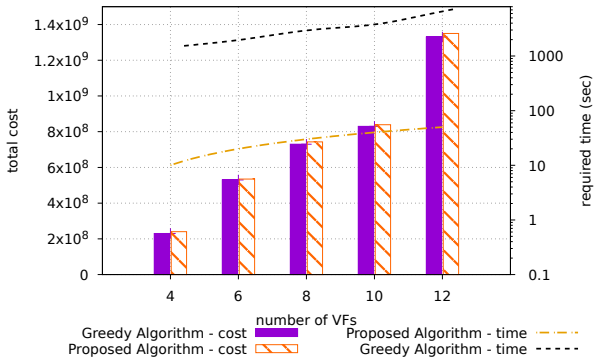

Fig. 2. Comparison between the optimum solution (greedy algorithm) and the proposed algorithm.

1) Solving Problem 1: If $t^* > \frac{1}{fps_{serv}}$, then the computational capacity of the edge devices is insufficient to support the service's QoS, if only one instance of each VF is deployed. In order to tackle this, we draw inspiration from the recently launched concept of Cloud-native functions [18], which handle dynamically the number of their instances aiming to handle the incoming requests. Thus, we propose a mechanism according to which a VF can be launched to multiple devices, and share the data coming from the previous VF of the VFC following a round-robin approach.

Thus, if we deploy a second instance of VF_j, the required time for the function to process the data related to a single

frame changes from $r_k(l)$ to $\frac{r_k(l)}{2} + b$, where b is the time overhead implied for handling the data separation on VF_{k-1} (previous VF in the chain) and data merging on VF_{k+1} (next VF in the chain), assuming that the two instances of the VF is deployed to identical nodes.

This case leads as to the formulation of a new sub-problem (*sub-problem C*). Its objective is to identify the minimum number of replicate instances for each VF that need to be deployed on the edge network, in order to meet the QoS constrains. Let $\overrightarrow{S} = [s_1, s_2, ..., s_m]$ represent the number of instances for each one of the VFs, with s_i being an integer larger or equal to one ($s_i \geq 1$). Thus, we derive the following problem formulation:

$$min \left\{ \sum_{j=1}^{m} s_j \right\} (13)$$

$$\sum_{j=1}^{m} \left(\frac{r_f(N_j)}{s_j} + (s_j - 1)b + \frac{output_j}{W_{ff'}} \right) \leq \frac{1}{fps_{serv}} (14)$$

, where N_j are the required instructions per frame required for executing VF_j on device f, with f' undertaking VF_{j+1}. (Eq. 13) drives our model to produce solutions with the minimum total new instances of the VFs, while (eq. 14) satisfies the QoS of the surveillance service. Unlike the problem formulated by *eq. (9-12)*, this is a non-linear mixed integer problem, which required the utilization of the active set solver APOPT [19]. Let the result of this sub-problem be $instances = [ins_1, ins_2, ..., ins_m]$. Using $instances$, we can revisit *Sub-problem B* and solve the allocation problem as before. The discrepancy of the latest described sub-problem is that the utilized nodes f and f' are unknown. This is rational, because we seek the number of the VF instances with regard to the computational time, which depends not only on the VF load but also on the node that will undertake the VF. We resolve this deviation by using Algorithm 1. This algorithm can provide two approaches: (a) worstCase scenario, where the edge device used to calculate (eq. 14) is the one with the lowest processing capacity and (b) bestCase scenario, where the highest processing capacity device is utilized.

Algorithm 1 receives as input the processing fps implied by the QoS and the allocation of the initial instances of the VFs. As depicted in Fig. 3 ('model' plots), both approaches converge to the desired processing fps. The reported results have been derived by a simulation framework that has been developed in order to evaluated the reported approach (github.com/blind-review- process). As far as the two functions ($addReplicate()$ and $removeReplicate()$) used in Algorithm 1, they calculate for each VF the improvement (for the $addReplicate()$) or the regression (for the $removeReplicate()$) a new instance will have to the total cost. Let $\{v_i\}$ be these values. Then, we choose the VF which minimizes the difference $|fps_{current} - v_i|$. In each iteration, *Sub-problem B* is solved. Aiming to evaluate the accuracy of the proposed algorithm for solving *Problem 1*,

Algorithm 1: Online placement optimization algorithm

Input: fps, x_{ij}, ϵ (error tolerance)
deploy(x_{ij});
$fps_{current} = measure_{fps}()$;
if $|fps_{serv} - fps_{current}| < \epsilon$ **then**
  return x_{ij}
else
  while $(|fps - fps_{current}| > \epsilon)$ **do**
    sleep(1);
    if $(fps_{real} < fps_{current} - \epsilon)$ **then**
      $x_{ij} = addReplicate(x_{ij})$
    else if $(fps_{real} > fps_{current} + \epsilon)$ **then**
      $x_{ij} = removeReplicate(x_{ij})$
    else
      return x_{ij}
    end
    deploy(x_{ij});
    $fps_{current} = measure_{fps}()$;
  end
end

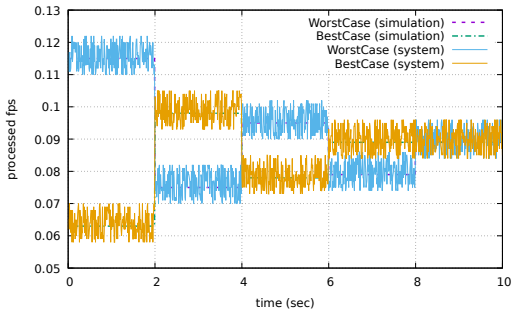

Fig. 3. Comparison between modelling and real-case system implementation.

a set of scenarios (edge networks and $VFCs$) have been setup, solving *Problem 1* both using a naive greedy algorithm (which calculates the optimum solution) and the proposed approach. Five different $VFCs$ have been used. For each VFC, 15 scenarios have been established by setting $cpu_{load} = 10^4 N(10, 0.8)$, $m_k = 10^3 N(20, 0.9)$, $Wtt' = 10^5 N(10, 0.7)$ and $output = 10^3 N(200, 0.65)$. Figure 2 presents the results of this comparison. The reported results are the average values of the total cost among the 15 different scenarios. The total average extra cost imposed by the proposed approach was $\approx 1.97\%$, while the required time for each algorithm to solve the problem was $\approx 5.61 sec$ for the proposed approach and $7.89 \times 10^3 sec$ for the greedy algorithm (Fig. 2). The solvers, which have been implemented using GEKKO suite [20], have been executed on a Intel i7 2.8GHz (8-core) on 8GB of RAM.

III. System Implementation

Aiming to evaluate the model described in Section II, both a simulation and a prototype edge network has been used, where all the necessary functionalities have been developed and de-

ployed, to support AI real-time video analytics of surveillance services. The simulation framework modeled each edge device as a Linux process. Linux commands *cpulimit* and *ulimit* were utilized to mock specific computational capacities for each 'device'. Each video analytic service has been modeled as a set of n VFs, where n is a random integer $\in [3, 5]$. Finally, each VF could by either a *light* VF, a *moderate* VF or a *heavy* VF, with relative computational characteristics each. Each VF may by identical with another VF with a probability of 15%, enabling the caching mechanism described in the following sections. Two different setups have been implemented. Setup I deploys a much simpler distribution model (as described in Section IV), while Setup II utilizes the VFC model with the caching mechanism. The implemented edge network comprise 6 Raspberry PI 3 (model B+) devices, with a Quad Core 64bit CPU @ 1.2GHz and 1GB RAM and 2 Raspberry PI 4 devices with a Quad core Cortex-A72 (ARM v8) 64-bit CPU @ 1.5GHz with 4GB RAM, running Raspbian OS. The feed from the camera was mocked as video file from the VIRAT dataset [21]. Two video analytics services have been deployed on the edge network. *Service A* and *Service B*, requiring gender and age classification respectively (pre trained deep learning models, based on [22]). Both *Service A* and *Service B* decompose to 4 VFs. *Service A* includes $VF_1()$, $VF_2()$, $VF_3()$ and $VF_4()$, while *Service B* includes $VF_1()$, $VF_2()$, $VF_5()$ and $VF_6()$. Details on the VFs: $VF_1()$: Frame acquisition and image enhancement (histogram equalization and Multi-scale retinex on low light conditions, $VF_2()$:Blob calculation for a specific frame, $VF_3()$: Convolutional Neural Network (MobileNet v2) pass and probability matrix acquisition, $VF_4()$: Coordinates calculation for the detected objects and non maxima suppression for overlapping objects, $VF_5()$: Convolutional Neural Network (gender CNN networks) pass and probability matrix acquisition, $VF_6()$: Results reporting. The model's basic parameters are: $n = 8$, $W = 100$ Mbps, $cpu_{load} = \left[10^3, 5 \times 10^3, 10^6, 10^2\right]$ instructions/frame, $output = \left[10^5, 3 \times 10^4, 10^7, 10^4\right]$ bytes, $c_k(l) = \frac{l^2+0.8}{1000}$, $r(l) = \frac{20l+9}{m_k}$ sec, $m_1 = 10^4$ instructions/sec(PI 3) and $m_2 = 10^7$ instructions/sec (PI 4). The values have been selected after performing a set of experiments for different workloads. A non-linear model for the cost function has been chosen. VFs have been implemented using Python3, utilizing the multiprocessing library.

IV. Results

One can argue that the VFC approach adds a lot of complexity on the management of the edge network, while the same results could be reached if multiple agents have been deployed on the network and each agent would execute the full stack of the VFs. It is obvious that this approach would only require one device to distribute the frames among the agents (Setup I). Yet, this approach, despite its simplicity, share two main drawbacks. The total cost of the edge network is greater than the VFC approach. In case of multiple services, there is no space for sharing the data among the services. The next section reports an analysis on these aspects, based on experiments held on the edge network described in the previous section. Fig. 5 and Fig. 4 present the experimental results of the simulation environment on Setups I and in II (Setup II refers to the proposed VFC model). It is obvious that the VFC model enables the support of a specific number of services with fewer edge devices and with substantially lower total cost, as the service demand scales. Regarding the real edge network, in order to support *Service A*, VFO deployed 1 instance of $VF1$, 2 of $VF2$, 4 of $VF3$ and 1 instance of $VF4$. For *Service B*, the resulted instances were 1, 3, 3 and 1 for $VF1$, $VF2$, $VF5$ and $VF6$ respectively. The calculated $VFCs$ are in accordance with the VF characteristics, as the most demanding VFs ($VF3$ and $VF5$) participated in the chains with the largest number of instances.

TABLE II
Experiments' setups.

Setup	Description	Remarks
Setup I	devices execute the whole stack of VFs	The network was setup using the approach described in Section II, assuming a VFC with only one VF.
Setup II	Proposed VFC model.	
Setup III	*Services A* (gender classification) and *B* (age classification) deployed on Virtual Chains. No caching between common VFs.	
Setup IV	*Services A* and *B* deployed on Virtual Chains. Caching mechanism deployed.	

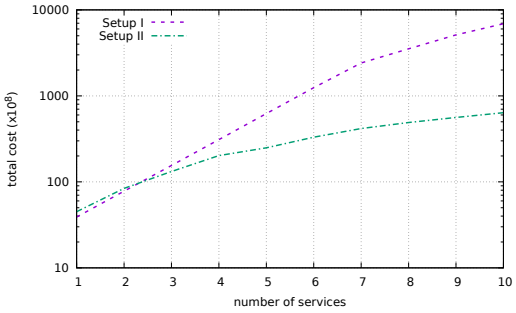

Fig. 4. Total network cost (simulation environment).

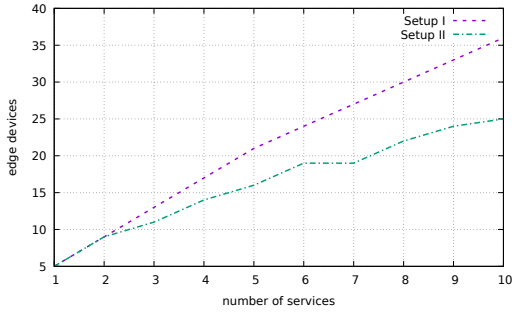

Fig. 5. Number of required edge devices for different number of deployed services (simulation environment).

A. Caching mechanism

Services A and *B* can share $VF1()$ and $VF2()$. For this, we have developed a caching mechanism, according to which when a node executes a VF for data related with frame k, it stores the results in a stack for a specific timeframe. In case another video analytics service request from the same VF to process data related to an already processed frame, it retrieves the results and forwards them to the next VF of the VFC, without recalculations. A set of experiments has been conducted, trying to reveal the benefits of the VFC approach. The results have been reported on Fig. 6 were collected after a 10 minute run of the system, for each setup. Four different setups were tested (Table II). Setups were configured in order to support the QoS constrains of *Services A* ($fps = \frac{1}{12}$) and *B* ($fps = \frac{1}{10}$). Fig. 6 illustrates that the QoS has been reached for all setups, i.e. the edge network was able to support all setups. When comparing though the total cost between Setups I and II, we notice that the VFC model outperforms the typical 'all-in-one' setup by saving approximately 51.2% of the cost. Finally, the caching mechanism tested in Setup IV decreases by 47.3% the total cost, when compared with Setup III.

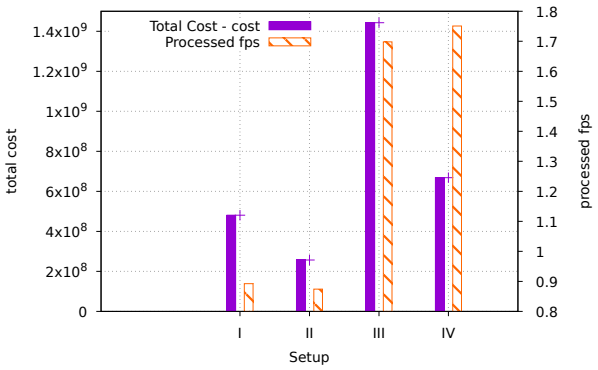

Fig. 6. Total network cost & processed fps for the different setups (real case environment).

V. Conclusions and Future Work

Edge computing is expected to be an important part of the AI industry during the next few years. Its advantages lie not only on the proximity with the processing data, but also on the data protection and safety issues, which are debatable on the Cloud computing paradigm. This paper proposes a novel concept for enabling real-time AI applications on an Edge network. Our proposal is based on the $VFCs$ which are used to distribute an AI application across the edge network on an scalable fashion. After providing a mathematical model for our system, we report the results of a real-case scenario, where the system has been implemented and tested in various setups. A caching mechanism is also described, which extents even further the capacity of our system. The experiments have provided evidence that such this approach can be used to undertake heavy-load AI applications and handle them real-time. Regarding the next step of our work, we plan to extent our model to have the capacity to handle node failures, by adding a migration mechanism to our architecture. Finally, a more extented comparison framework will be developed, aiming to compare the performance of the proposed model against general distribution frameworks like Apache Spark©.

Acknowledgment

This is work is partially funded by the Innovate UK Open Round 2 program: Wide Smart Safe, Robust and Resilient Smart Cities Application Using Fog Computing (WATCH, TSB-103845) project.

References

[1] Jain et al., "Performance characterization of dnn training using tensorflow and pytorch on modern clusters," in *Proc. of the 2019 CLUSTER*. 2019, IEEE.
[2] He et al., "A Survey to Predict the Trend of AI-able Server Evolution in the Cloud," *Special Section on Emerging Trends, Issues and Challenges in energy - efficient Cloud Computing*, vol. 6, Feb. 2018.
[3] Sun et al., "Vu: Edge computing-enabled video usefulness detection and its application in large-scale video surveillance systems," *IEEE Internet of Things Journal*, 2019.
[4] Rob Kitchin, "The real-time city? Big data and smart urbanism.," *GeoJournal*, vol. 79, no. 1, pp. 1–14, Feb. 2014.
[5] W. Zhou et al., "A System Architecture to Aggregate Video Surveillance Data in Smart Cities," in *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–7.
[6] Li, Jianhua et al., "Virtual Fog: A Virtualization Enabled Fog Computing Framework for Internet of Things," *IEEE Internet of Things Journal*, vol. 5, no. 1, Feb. 2018.
[7] Chen et al., "Dynamic Urban Surveillance Video Stream Processing Using Fog Computing," in *2016 IEEE Second International Conference on Multimedia Big Data (BigMM)*, 2016, pp. 105–112.
[8] Dautov et al., "Metropolitan intelligent surveillance systems for urban areas by harnessing IoT and edge computing paradigms," *Softw Pract Expe.*, vol. 48, no. 1, pp. 1475–1492, 2018.
[9] M. Satyanarayanan, "The Emergence of Edge Computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2019.
[10] Ning Chen and Yu Chen, "Smart City Surveillance at the Network Edge in the Era of IoT: Opportunities and Challenges," in *Smart Cities*, pp. 153–176. Apr. 2018.
[11] Chen et al., "Distributed deep learning model for intelligent video surveillance systems with edge computing.," *IEEE Transactions on Industrial Informatics.*, 2019.
[12] Puthal et al., "Secure authentication and load balancing of distributed edge datacenters," *Journal of Parallel and Distributed Computing*, vol. 124, pp. 60–69, 2019.
[13] Hwejoo et al., "A data streaming performance evaluation using resource constrained edge device," in *2017 ICTC*, Jeju, South Korea, 2017, IEEE.
[14] J. Halpern and C. Pignataro, "RFC 7665 - Service Function Chaining (SFC) Architecture," Oct. 2015.
[15] C. Besse and B. Chaib-draa, "An Efficient Model for Dynamic and Constrained Resource Allocation Problems," in *2nd COPLAS '07*, 2007.
[16] Ahuja, Ravindra et al., *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, 1993.
[17] Laborie, P. et al., "IBM ILOG CP optimizer for scheduling," *Constraints*, vol. 23, no. 2, pp. 210–250, 2018.
[18] Aderaldo et al., "Kubow: an architecture-based self-adaptation service for cloud native applications," in *Proc. of the 13th ECSA*, Paris, France, 2019, vol. 2, pp. 42–45, ACM.
[19] Hedengren et al., "Apopt: Minlp solver for differential and algebraic systems with benchmark testing," in *Proceedings of INFORMS National Meeting*, 2012.
[20] Beal et al., "Gekko optimization suite," *Processes*, vol. 6, no. 8, 2018.
[21] Sangmin et al., "A Large-scale Benchmark Dataset for Event Recognition in Surveillance Video," in *Proceedings of IEEE Comptuer Vision and Pattern Recognition (CVPR), 2011*. 2011, IEEE.
[22] Google AI Blog, "MobileNetV2: The Next Generation of On-Device Computer Vision Networks," 2018.