Scheduling Massive Camera Streams to Optimize Large-scale Live Video Analytics

Chenghao Rong, Jessie Hui Wang*, Juncai Liu, Jilong Wang, Fenghua Li, and Xiaolei Huang

***Abstract*—In smart cities, more and more government departments will make use of live analytics of videos from surveillance cameras in their tasks, such as vehicle traffic monitoring and criminal detection. Obviously, it is costly for each individual department to deploy its own infrastructure, *i.e.*, cameras and analytics system. In this paper, we consider a scenario in which a city deploys an infrastructure and departments submit requests to access and analyze videos for their own purposes. The live analytics of massive streams is computation-intensive and the tasks might be latency-critical, which makes scheduling massive streams to optimize all tasks an essential and challenging work. We exploit an end-edge-cloud architecture and propose an adaptive system to schedule the massive camera streams and tasks, which considers all factors affecting the computation and networking resource consumption, *e.g.*, sharing of model computation, video quality, model partition, and task placement. Particularly, the resource consumption of *Faster R-CNN + ResNet101* under each partition scheme is profiled for the first time and we notice the partition must be used together with lossless compression techniques to be beneficial. Furthermore, sometimes tasks might be required to migrate because the scheduling decision made by the system changes to adapt to the changing resource supply and demand. In order to avoid the performance degradation during migration, we propose a non-destructive migration scheme and implement it in the system. Simulations demonstrate our system achieves a total utility close to the maximum and our analytics system performs better than state-of-the-art solutions.**

I. Introduction

Surveillance cameras are being more and more widely deployed. It is reported that nearly 1 billion cameras have been installed worldwide, and there has been one camera for every 4.1 people in China [1]. These cameras continuously produce videos that are potentially informative for many government departments and economic entities. Hopefully, in future smart cities, one city can deploy cameras in public areas and establish computation facilities to analyze these videos. Then departments and entities do not need to deploy their own infrastructures, and they can just submit their requests to access and analyze the videos from their interested cameras for their purposes, *e.g.*, traffic control, AMBER Alerts, and smart retailing.

Let us define a request to access and analyze a video stream from a single camera for a particular purpose as a *video analytics task*. Obviously, a single camera stream can be used for multiple tasks simultaneously, and there are massive camera streams. The tasks may have different performance requirements. For example, some tasks can be latency-critical, and some tasks prefer higher accuracy to shorter latency.

Jessie Hui Wang is the corresponding author (jessiewang@tsinghua.edu.cn).

The live analytics of a stream requires intensive computation resources that cheap cameras widely deployed in smart city usually do not have. A cloud may have sufficient computation resources, but transmitting video streams to the cloud introduces extra end-to-end delays. Nowadays, camera-edge-cloud architecture has been regarded as the only feasible approach for massive video stream analytics [2]–[5]. In the camera-edge-cloud architecture, people deploy edge nodes and also deploy a cloud with more abundant computation resources than each edge node. An edge node is responsible for multiple nearby cameras. Intuitively, latency-critical tasks can be placed on edge nodes nearby the corresponding camera to enjoy shorter transmission delay, and accuracy-sensitive tasks can be placed in the cloud to enjoy more computation resources for higher accuracy.

However, large-scale live analytics of video streams from massive cameras in the camera-edge-cloud architecture faces many difficulties. Considering the massive camera streams and tasks, the available computation resource of edge nodes and the wide-area networking resource from each edge node to the cloud are likely to be insufficient. Meanwhile, the system's environment is dynamic. The wide-area network bandwidth between an edge node and the cloud is highly variable [6], and users may submit or terminate tasks during the video analytics system running. Obviously, as the resource supply and demand changes, the optimal scheduling decision should change accordingly. In order to optimize the analytics performance, we have to *design an adaptive system to schedule these camera streams and tasks carefully to use the resources efficiently*, which is the problem to be solved in this paper.

There have been some works on making proper decisions to optimize video analytics performance, but most of them do not exploit end-edge-cloud architecture because they only consider a single stream or multiple streams with a single performance metrics and then an end-edge architecture or end-cloud architecture has been sufficient to satisfy their needs [6]–[16]. In a scenario with massive streams and a lot of tasks with diverse performance requirements, an end-edge-cloud architecture is a necessity, and it results in a more complicated objective function, more scheduling dimensions, and more resource constraints.

In a large-scale video analytics system, available resources are shared by all camera streams and tasks. The first problem in achieving optimal performance is to avoid redundant computation and data transmission to make sure resources are used efficiently. As mentioned above, there can be multiple tasks associated with the same camera stream. It is possible for these

tasks to share the model computation since the state-of-the-art DNN can detect multiple types of objects at the same time. In this paper, *we design the system framework in a way that enables sharing of model computation as much as possible*, in which a stream can have two *pipelines*, one for a small model (*e.g.*, *Faster R-CNN + ResNet50* [17]) on an edge node, and one for a big model (*e.g.*, *Faster R-CNN + ResNet101*) in the cloud. Tasks associated with the same camera stream and placed on the same place can share model computation.

The second problem is to find the optimal decisions according to the tasks' performance preferences and available resources. In this work, we try to consider all available decision dimensions that have influence on the resource consumption and performance achievement of tasks. Besides video quality (frame rate and resolution), object detection model and task placement (edge or cloud), we pay special attention to Deep Neural Networks (DNN) partition schemes. *DNN Partition* was proposed to trade a small amount of edge computation resource for wide-area networking resource, but whether a model should be partitioned and where to partition should be carefully answered. Previous works [18], [19] conducted DNN partition profiling only for small models with dozens of layers, such as AlexNet, perhaps because their goals are to help mobile devices which can only handle small models. In our scenario, the cloud has sufficient computation resource to exploit complex models to achieve better accuracy. Therefore, we conduct DNN partition profiling for a complex model *Faster R-CNN + ResNet101*, which is one of the state-of-the-art object detection model. We find that *the profiling result of this complex model is different from small models, and compressing feature maps is necessary to make the partition of this complex model useful*.

The third problem is to ensure the performance of tasks during migration. The system environment is dynamic, which means the scheduling decision can change anytime. Once the scheduling decision changes, some tasks might be required to migrate from cloud to edge or vice versa. A simple "stop-and-start" migration scheme would cause performance degradation of the migrated tasks, and we have not seen any research work on non-destructive migration solutions. In this work, we comprehensively analyze the possible reasons for performance degradation during migration, and design a non-destructive migration scheme. Our migration scheme can minimize the negative influences of migrations by warming up the migrated tasks in the destination place as much as possible and minimizing the amount of data which is transmitted from the source place.

In summary, we make the following contributions in this work.

- We conduct DNN partition profiling for a complex state-of-the-art object detection model, *i.e.*, Faster R-CNN + ResNet101, and report important observations which have never been reported before. For this model, we must compress in-layer feature maps to make its DNN partition useful for system efficiency improvement.
- We formulate and solve the scheduling decision problem to maximize the total utility of tasks with diverse performance requirements for a video analytics system in a smart-city scenario, in which the deployed infrastructure is with limited computation resources and varying bandwidth between edge nodes and the cloud. Evaluations demonstrate that our system achieves a total utility close to the maximum and our system performs better than state-of-the-art solutions.
- We comprehensively analyze the possible reasons for performance degradation during migration, and design a non-destructive migration scheme to avoid the performance degradation of tasks. Evaluations demonstrate that our migration scheme enables a near-seamless migration without perceptible impact on system computation load and task latency.

We implement a system which incorporates all the above considerations and mechanisms, and make the source code publicly accessible [20]. The remainder of this paper is organized as follows. In Section II, we review the related work. In Section III, we discuss the system architecture and the specifications of video analytics pipeline. In Section IV, we report our works and observations on the profiling of *Faster R-CNN + ResNet101*. In Section V, we formulate the scheduling problem, and a heuristic algorithm is proposed to solve the optimization problem. In Section VI, we propose a non-destructive migration scheme to avoid the performance degradation of the migrated tasks. We evaluate our proposed system in Section VII. Section VIII concludes the paper.

II. Related Work

Profiling for DNN Partition. DNN partition profiling must be conducted in advance before DNN partition is used for resource scheduling. During profiling, the resource consumption under different partition schemes of the model under study is measured. Previous works such as [18], [19], [21]–[23] conducted DNN partition profiling only for small models with dozens of layers, such as AlexNet, VGG, and YOLOv2. In this work, we conduct profiling for Faster R-CNN + ResNet101, which is never profiled in previous works. Some important observations that are different from the profiling results of small models are reported.

Video Analytics System. There have been many works on optimizing the performance of analyzing a single video stream [7]–[10], [24]–[32]. Their solutions cannot solve the problem in our scenario because resource contention among massive video streams is not considered.

Some works focused on multiple-stream scenarios and considered the resource contention among streams, but they scheduled only one type of resources, either computation or communication [6], [11]–[15], [33]–[36]. In [11]–[14], [33], [34], they assumed that all cameras are close to the server (cluster) that conducts analytics, and they had sufficient networking bandwidth to transmit camera streams to the server, so they only considered the allocation of computation resources. In [35], [36], they proposed a framework in which edge

nodes extract RoIs (region of interests) in frames and these RoIs instead of the original frames are sent to the cloud for accurate analytics. They just tried to reduce networking resource consumption but did not consider the competition on networking resources among streams, so they only considered computation resources when making scheduling decisions. In [6], [15], videos were sent to the cloud over the Internet, therefore networking resources should be considered during scheduling. They only used the cloud to conduct analytics directly, so they assumed that computation resources are sufficient. In our scenario, multiple cameras compete for the computation resources on an edge node, and videos are sent to the cloud over the Internet, so both computation and networking resources should be bottlenecks.

The most recent works have noticed that both computation and networking resources can be bottlenecks, but they have different design goals [5], [21], [37]. We are optimizing the total utility of tasks with diverse performance requirements. In [37], the authors focused on a market issue, *i.e.*, solve the conflicts between edge operators and cloud operators and maximized the social welfare. [21] aimed to minimize latency for tasks. The authors of [5] aimed to optimize the accuracy of tasks.

In [38], the authors provided a solution for a similar scenario using a framework different from us. It includes multiple candidate analytics models and aims to optimize the total QoS (quality of service) of all tasks by making optimal decisions on model selection and placement for each task. In our work, only Faster R-CNN + ResNet is provided. Faster R-CNN + ResNet has been the most widely used model in object detection [12], [32], and using the same model is beneficial to achieve more efficient resource usage. Meanwhile, we also use DNN partition to reduce network bandwidth demands, which can help to solve the challenge of scarce networking resource between edge nodes and a cloud.

Migration of Video Analytics Task. None of these live video analytics systems proposed solutions to mitigate the performance degradation of analytics tasks during migrations caused by scheduling decision changes. Virtual machine (VM) migration (or container migration) in datacenter networks have been studied by a lot of researchers such as pre-copy [39], post-copy [40], and hybrid-copy [41]. However, these research works cannot be directly applied to our problem. The downtime of these migration techniques might be too long in our scenario because the available networking bandwidth in data centers is usually much larger than the WAN bandwidth between an edge node and the cloud in our scenario. Furthermore, these migration techniques have not been optimized based on the characteristics of the video analytics tasks so that the amount of data to be transmitted during migrations is huge, which causes an excessive latency of a migrated task.

III. Overview of System

In the section, we introduce the problem scenario, the concept of *video analytics pipeline*, and the framework of our system.

A. Problem Scenario and Concept of Video Analytics Pipeline

The system works in a camera-edge-cloud architecture which is composed of cameras, edge nodes, and a cloud. Each edge node can serve multiple cameras and it can communicate with the cloud to send video streams to the cloud for analytics. A single camera stream can be used for multiple tasks simultaneously for their own purposes. Available computation and networking resources are shared by all tasks associated with camera streams. We have to schedule these camera streams and tasks carefully to optimize the analytics system performance.

A typical video analytics task contains many *components*. Figure 1 shows an example of a single camera stream, and the video stream of this camera is used by several tasks, *i.e.*, pedestrian violation detection, car counter, and auto-driving assistance service. These tasks may focus on different types of objects, say pedestrians, cars, or trucks. The object detection is computation-intensive. Fortunately, the state-of-the-art DNN can detect multiple types of objects at the same time, therefore it is possible for these tasks to share the results of object detection and retrieve their interested objects from the results. In other words, we do not need to run an instance of the object detection model for each task.

However, the tasks associated with a single camera stream may have different performance requirements that cannot be satisfied by a single model instance. Taking Figure 1 as an example, the auto-driving assistance service (for non-critical purposes) is more sensitive to latency, while the pedestrian violation detection prefers a better accuracy. In case that the resource allows, the auto-driving service may prefer to see the object detection model being placed on the edge node which is close to the camera, while the pedestrian violation detection would like to see the object detection model being placed in the cloud which can achieve high accuracy.

We define the workflow of the model instance to analyze a camera stream as a *video analytics pipeline*. The camera stream shown in Figure 1 has two pipelines. One pipeline is deployed for the tasks that consider latency as more important than accuracy. It runs a small object detection model (*e.g.*, *Faster R-CNN + ResNet50*) on an edge node, and thus it is referred to as *edge pipeline*. Small models incur smaller latency and demand less computation resources than big models, which makes them more suitable for edge nodes. The other pipeline is deployed for the tasks that prefer accuracy. It is deployed in the cloud and uses big models (*e.g.*, *Faster R-CNN + ResNet101*), thus we refer to it as *cloud pipeline*. In case that DNN partition technique is used, some earlier layers of the big model of a cloud pipeline can be executed by edge nodes.

B. System Framework

The framework of our system is shown in Figure 2. Our system consists of a centralized *system manager* and distributed *machine managers*.

The system manager is located in the cloud and it is responsible for making scheduling decisions, *i.e.*, allocating available resources to streams. There are three key modules:

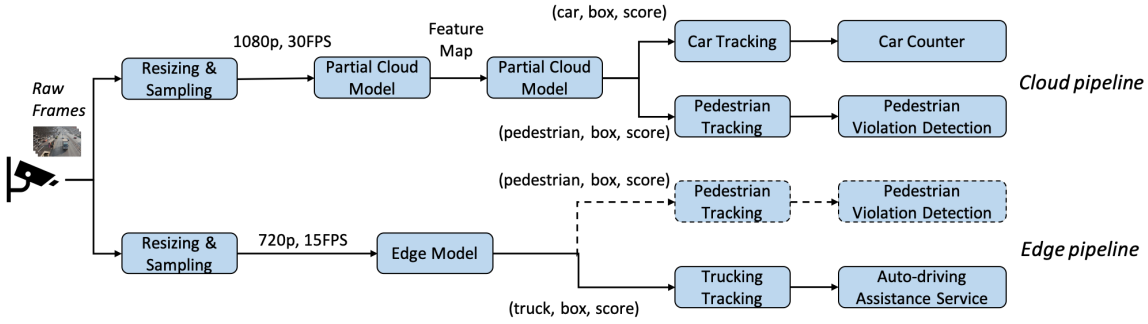

Fig. 1. An example to illustrate a single camera stream.

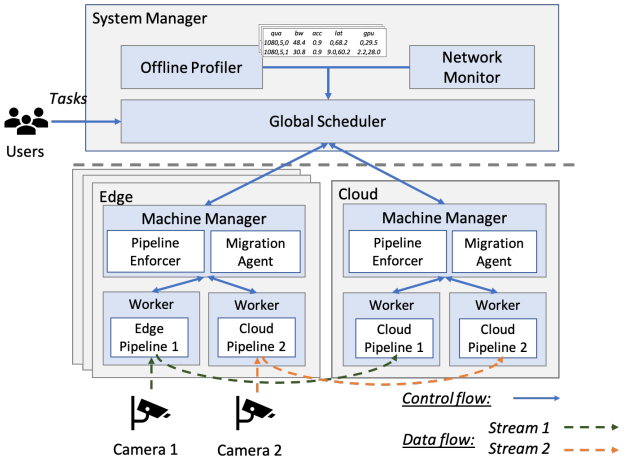

Fig. 2. The framework of our system.

Offline Profiler, *Network Monitor*, and *Global Scheduler*. The offline profiler is used to learn task performance and resource consumption under various configurations (such as video quality, object detection model, and partition scheme) in each edge node and the cloud. The network monitor is responsible for periodically monitoring network bandwidth from each edge node and the cloud. Users submit video analytics tasks with various utility functions (performance preferences) to the global scheduler. The global scheduler is responsible for making the optimal scheduling decisions to maximize the total utility of all tasks according to the profiling results, the result of the network monitor and utility functions. Then, the scheduler distributes its scheduling decision to the machine managers of edge nodes and cloud. In case that the global scheduler decides that a task should be migrated, it is also responsible for sending a migration request to the machine manager of each involved party.

Each edge node and the cloud has its own machine manager, which consists of two modules. *Pipeline Enforcer* is responsible for enforcing scheduling decisions made by the global scheduler, such as tuning parameters, starting new pipelines and stopping unneeded pipelines. Each pipeline is executed in a pre-built docker container to ensure the security and isolation. *Migration Agent* is responsible for the communication between the source node and the destination node during a migration to complete task migrations (more in Section VI).

C. Describing Scheduling Decisions

A scheduling decision made by the global scheduler should be distributed to all machine managers, therefore we should develop a way to describe a scheduling decision to make sure that machine managers can understand the scheduling decision.

Figure 3 shows the key part of a file that describes an edge pipeline. The `camera_id` specifies which camera stream this pipeline is associated with. The `name` is used to identify this pipeline. The `task_id` field specifies which tasks are using this pipeline.

The `components` field contains the information about all components within this pipeline. Each component is tagged with an `id` for identification. The `name` field describes the function of the component. The `input_component_id` field specifies the component who sends data stream to this component. Generally said, a "decoder" component is usually the source component of a pipeline, so it has no input_component_id.

A `args` field specifies the key parameters of the component at run-time, and its composition varies from component to component. There are three fields that are essential for describing a scheduling decision. `output_resolution` and `output_framerate` of a "decoder" component specify the video quality configuration of this pipeline selected by the global scheduler. `partition` of a "object detection" component specifies the partition scheme of this pipeline, in which "null" means the model is not partitioned.

IV. DNN Partition Profiling for Faster R-CNN + ResNet101

DNN partition has been proposed in some works, *e.g.* [18], [19], to achieve their particular performance goals, *e.g.* latency minimization or energy conservation. With DNN partition, a DNN model can be split into two parts, running on two different locations, *e.g.*, one on an edge node, and the other in the cloud. The feature map of each frame output by the last layer of the head part, instead of the original frame, needs

```
    "camera_id": 0,
    "name": "edge_pipeline",
    "task_id": [0, 1],
    "components": [
        {"name": "decoder",
         "id": 0,
         "args": {
            "camera_ip": "203.91.121.212",
            "port": 5000,
            "output_resolution": "1080p",
            "output_framerate": 10 } },
        {"name": "object detection",
         "id": 1,
         "input_component_id": 0,
         "args": {
            "memory": 0.1,
            "model_path": "edge/frozen_inference_graph.pb",
            "label_path": "edge/label_map.pbtxt",
            "partition": "null" } },
        {"name": "object tracking",
         "id": 2,
         "input_component_id": 1,
         "args": {
            "threshold": 0.1,
            "object_class": "car" } },
        {"name": "object tracking",
         "id": 3,
         "input_component_id": 1,
         "args": {
            "threshold": 0.1,
            "object_class": "pedestrians" } },
```

Fig. 3. An example of describing a video analytics pipeline.

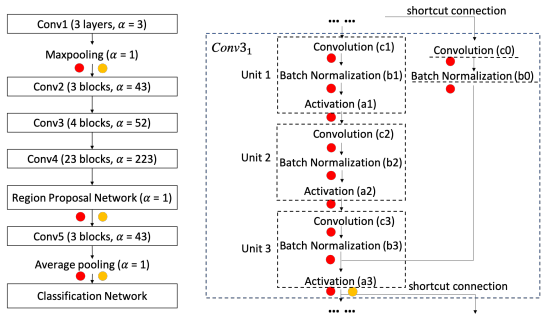

Fig. 4. The structure of *Faster R-CNN + ResNet101*. The colored dots represent cut points.

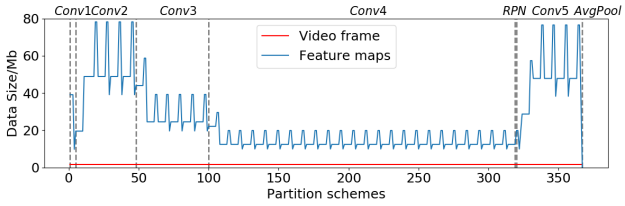

Fig. 5. Partition profiling results of *Faster R-CNN + ResNet101* (1920x1080).

to be transmitted to the tail part to complete the inference task, which consumes the networking resource between the two places.

Obviously, the partition scheme (*i.e.*, where to split the model) determines the amount of resources that the DNN model needs to consume, including the computation resource on each of the two places and the networking resource between the two places. When using DNN partition, we have to measure the computation and networking demand of a DNN model with different partition schemes. In this section, we will conduct DNN partition profiling for *Faster R-CNN + ResNet101*, which is never profiled in previous works. The other difference is that we need to profile one more element, the *GPU utilization*, besides the two elements profiled by previous works, *latency* and the *data size*. In our scenario, a lot of model instances are competing for the computation resource of edge nodes and the cloud, therefore we have to know the computation demand of each model in terms of GPU utilization rate. In previous works, there is no resource competition because they only consider a single task, so they only need to profile the inference time and data size (which also affects latency).

A. The Structure and Partition Schemes

Figure 4 shows the structure of *Faster R-CNN + ResNet101*. In the model, the *Faster R-CNN* part includes a Region Proposal Network (RPN) and a Classification Network, and the *ResNet101* part consists of 5 components, namely from *Conv1* to *Conv5*, and there are 33 blocks in these components. There are 3 units in each block, and each unit contains three layers, *i.e.*, a convolution layer, a batch normalization layer, and an activation layer. Each block is designed to be with a branchy structure, *i.e.*, there is a *shortcut connection* connecting two adjacent blocks [42].

Potentially, the model can be split between any two neighboring layers, and a scheme may result in one cut point or two cut points (when splitting the model in a place with a shortcut connection). Taking the first block of *Conv3* (denoted by $Conv3_1$) shown in the second subplot of Figure 4 as an example. It has 9 layers in its backbone connection and 2 layers in its shortcut connection. In total, there are $8 \times 2 + 1 = 17$ partition schemes in this block.

Let α denote the number of potential partition schemes. We show α for each component in Figure 4. Besides splitting the model within five components, we also consider the partition schemes that split the model after the max pooling layer, the average pooling layer, and the RPN. In total, we conduct measurements for 367 partition schemes.

For convenience, we use the name of the last layer of the head part to represent the partition scheme. The layer name is shortened as the second subplot of Figure 4 indicates, *e.g.*, *c0* means the convolution layer of *shortcut connection*.

B. Partition Profiling of Faster R-CNN + ResNet101

A partition scheme is usable only when it incurs less networking demand than the original video stream. Therefore, we first focus on the data size of a transmitted feature map under various partition schemes. The profiling results for images with a resolution of 1920x1080 is plotted in Figure 5. Surprisingly, we have the following observation from the results.

Observation: The in-layer feature map that is needed to be transmitted across the wide-area network is much larger than the original frame under almost all partition schemes.

The only exception is the average pooling layer (the last partition scheme in Figure 5), which is almost at the end of

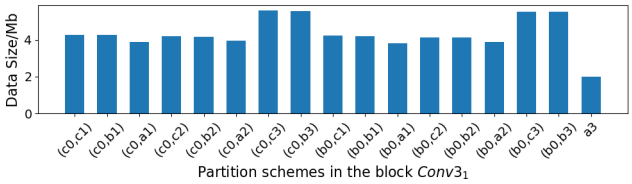

Fig. 6. The data size of compressed feature maps in the block $Conv3_1$

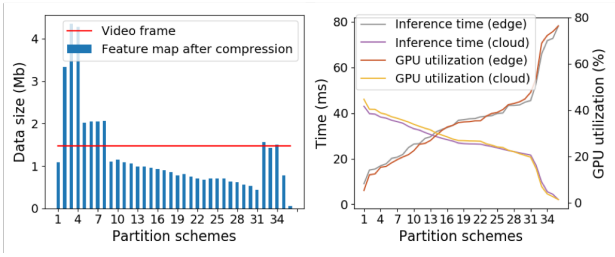

Fig. 7. The profiling results of DNN partition with compression when the video quality is 1080p, 15fps.

the model. However, if we split the model after this layer, the processing time of the head part in cloud is 61.52ms, accounting for 94.50% of the total processing time. It means this partition scheme significantly increases the computation overhead of edge nodes. The results for other resolutions are skipped as they are similar to Figure 5.

We look into the previous works and find that their selected partition scheme is also at the end part of model, such as *pool5* in AlexNet [18]. But *fc6* (the next layer of *pool5*) consumes more than 30% of the total computation resource consumption. Therefore, offloading a very small number of layers to the cloud can help a lot in their scenarios, while it is not true for the *Faster R-CNN+ResNet101*.

Fortunately, our edge servers are much more powerful than mobile devices. Compressing and decompressing feature maps on edge servers and the cloud would only consume negligible computation resource and induce negligible latency compared to the DNN inference task. Therefore, we conduct more measurements to see if compressing feature maps to reduce the size of feature maps can give us usable partition schemes.

C. Profiling of Partition + Compression

1) Compression Method: The compression of feature maps must be lossless to maintain the accuracy of object detection. A feature map is consist of 32-bit floating-point numbers. The lossless compression algorithms for floating-point numbers usually provide 1.5x-4x data reduction [43], [44], which cannot meet our requirements. The lossless compression of integers usually performs much better, therefore we first convert 32-bits floating-point feature maps to 8-bits quantized feature maps using the method in [45]. Recent studies have demonstrated that the uniform 8-bits quantization has a negligible effect on the accuracy of object detection [45].

Now we can compress quantized feature maps using lossless compression methods for integers. A quantized feature map includes many channels, and each channel is a 2D integer matrix. We tile the channels to convert the 3D matrix into a giant 2D matrix, then use the PNG to compress the 2D matrix.

2) Profiling Results of Partition + Compression: Now we take a look at the data size of a feature map after compression. Figure 6 show the data size for the partition schemes within the block $Conv3_1$. We can see that the last partition scheme, which is to split the model after the last layer of the block (the only partition scheme with a single cut point) produces the smallest bandwidth demand after compression. Other blocks have the same observation.

We notice that the above observation is not true before compression. [19] shows that the partition schemes with two cut points can produce the smaller data size in some models, *e.g.*, *AlexNet*, *Googlenet*. We conduct more measurements and find that *the feature maps after activation layers have more zeros and are more concentrated*, which help the last partition scheme achieve the smallest data size after compression.

Furthermore, we find that a single block only consumes a small proportion (about $0.7\% - 6\%$) of the total computation resource demanded by the whole model. It means the partition schemes within one block result in comparable computation overhead and the last scheme among them produces smallest networking demand. Therefore, for each of the 33 block, we only select the last partition scheme as a candidate. Together with the schemes after *maxpooling* layer, after *RPN* and after *average pooling* layer, we select 36 schemes to profile.

The profiling results for videos with a configuration 1080p, 15fps are plotted in Figure 7. We can see that compressing feature maps makes some partition schemes consuming less networking resource than an original frame with H.264 encoder, and allows DNN partition to be used. The profiling should be done for each video quality configuration. We find that there is no usable schemes for some video qualities even after compression. For example, under the video quality of (360p, 15fps), all partition schemes have a larger networking resource demand than the video encoded using h.264 (about 7.84Mbps). The schemes that have larger demand size than the video encoded using h.264 are filtered and will not be considered during scheduling.

Due to page limitation, we only introduce our work on DNN partition profiling and skip the details on task profiling and model profiling.

V. Scheduling Video Streams for Massive Tasks

In this section, we analyze the problem of scheduling video streams for massive tasks and propose a feasible solution for the problem. We start from analyzing the scheduling problem of a single video stream to get some insight for the complex scheduling problem of multiple video streams.

A. Analyzing a Single Camera Stream

As mentioned above, a video stream can serve multiple tasks with different performance requirements. We would like to see these tasks can share model computation, *i.e.*, using the

same pipeline. But whether multiple tasks agree to share the same model instance and where to place the model instance depend on a lot of factors, such as the available networking resource, the computation resource supply on edge nodes and the cloud, the performance requirements of tasks, and the accuracy-latency-demand profiling of various object types.

Let us summarize all possible scheduling plans for a camera stream and calculate the total utility of all tasks associated with this stream under each possible scheduling plan. There are the following decision variables for the scheduling problem of a single stream.

- For the edge pipeline, we can make decision on its frame rate and its frame resolution.
- For the cloud pipeline, we can make decision on its frame rate, its frame resolution and the partition scheme of the model instance.
- For each task associated with this camera, we can make decision on which pipeline it uses.

Therefore, theoretically, the number of the combinations of the above decision variables is $|\text{frame rate}|^2 \times |\text{frame resolution}|^2 \times |\text{partition scheme}| \times 2^{\mathcal{K}_i}$, where $\mathcal{K}_i$ is the number of tasks associated with the camera i, and $|\text{frame rate}|$ represents the number of frame rates that can be selected with available resources, and so on. We would like to emphasize that it is unnecessary to deploy two pipelines for a single stream on the same location (edge node or cloud). For any type of objects, the performance of the detection model always increases with the video quality. Therefore, a single pipeline with the best video quality allowed by the available resources has been enough for all tasks deployed on the same location.

In our system, we implement frame queues to temporarily store frames that are not analyzed in time. To ensure there are no ever-increasing frame queue, the inference latency must be less than the interval between two consecutive frames, *i.e.*, the reciprocal of the selected frame rate. Let $\mathcal{G}_i$ denote the set of possible scheduling plans for the stream of the camera i. We must ensure the following conditions hold for each element $\mathcal{G}_{i,j}$ in $\mathcal{G}_i$,

$$\begin{aligned} \forall j : \tilde{l}_i(\mathcal{G}_{i,j}) &\leq 1/\mathcal{G}^f_{i,j} \\ \forall j : \bar{l}_i(\mathcal{G}_{i,j}) &\leq 1/\mathcal{G}^{\bar{f}}_{i,j} \\ \forall j : \hat{l}_i(\mathcal{G}_{i,j}) &\leq 1/\mathcal{G}^{\bar{f}}_{i,j} \end{aligned}$$

Here, $\mathcal{G}^f_{i,j}$ and $\mathcal{G}^{\bar{f}}_{i,j}$ are the frame rate of the edge pipeline and the cloud pipeline respectively in the scheduling plan $\mathcal{G}_{i,j}$; $\tilde{l}_i$ is the inference latency of the edge pipeline, $\bar{l}_i$ is the inference latency of the earlier part of the detection model on the edge node (it can be zero when the whole model is in the cloud), and $\hat{l}_i$ is the inference latency of the later part of the detection model in the cloud. $l_i(\mathcal{G}_{i,j})$, $\bar{l}_i(\mathcal{G}_{i,j})$ and $\hat{l}_i(\mathcal{G}_{i,j})$ are given by the profiling results of the detection model on the same GPU type as the edge node and the cloud.

Let us assume we have $\mathcal{N}_i$ feasible scheduling plans for the camera stream i after removing the combinations of decision variables that cannot satisfy the above conditions. The total utility of all tasks associated with the camera stream i under $\mathcal{G}_{i,j} (j \in [1, \mathcal{N}_i])$, denoted by $\mathcal{U}_{i,j}$, can be calculated as follows.

$$\mathcal{U}_{i,j} = \sum_{k=1}^{\mathcal{K}_i} u_i^k(a_i^k(\mathcal{G}_{i,j}), l_i^k(\mathcal{G}_{i,j})) \tag{1}$$

Here, u_i^k is specified by the kth task associated with the ith camera. Roughly speaking, it should be a function of the latency and accuracy the task can achieve, *i.e.*, a_i^k and l_i^k, which are the results yielded by the selected scheduling plan $\mathcal{G}_{i,j}$. Please note the functions $a_i^k(\mathcal{G}_{i,j})$ and $l_i^k(\mathcal{G}_{i,j})$ are learned by the profiling of the task under various scheduling plans.

B. Scheduling Multiple Video Streams

Now let us consider n cameras in the system. Assume there are m edge nodes in the system, denoted by $S = (s_1, ..., s_m)$. Each camera $i \in [1, n]$ is connected to its corresponding edge node $s(i)$, and an edge node is providing services for multiple cameras.

The objective is to maximize the total utility of all tasks associated with these n cameras by selecting proper scheduling plans for every camera stream i from its set of possible scheduling plans $\mathcal{G}_i$. Mathematically, we are trying to determine $x_{i,j}$ $(j \in [1, \mathcal{N}_i])$ for every i, that satisfies the following conditions,

$$\max_{x_{i,j}} \quad \sum_{i=1}^{n} \sum_{j=1}^{\mathcal{N}_i} \mathcal{U}_{i,j} \cdot x_{i,j}, \tag{2}$$

$$\forall i : \sum_j x_{i,j} = 1, \tag{3}$$

$$x_{i,j} \in \{0, 1\}, \tag{4}$$

Here, $x_{i,j} = 1$ means that $\mathcal{G}_{i,j}$ is selected for the camera i, and $\mathcal{U}_{i,j}$ is the total utility of tasks associated with the camera i given that $\mathcal{G}_{i,j}$ is selected. Equation 3 ensures that one scheduling plan is selected for one camera. Equation 2 is the objective function of our scheduling problem.

The optimization problem should be solved subject to some feasibility conditions. In terms of the computation resource on edge nodes and the cloud, we should have

$$\sum_{i=1}^{n} \sum_{j}^{\mathcal{N}_i} r_i(\mathcal{G}_{i,j}) \cdot x_{i,j} \leq \mathcal{R}, \tag{5}$$

$$\forall t : \sum_{\{i|s(i)=t\}} \sum_{j}^{\mathcal{N}_i} d_i(\mathcal{G}_{i,j}) \cdot x_{i,j} \leq \mathcal{D}_t, \tag{6}$$

Here, $\mathcal{R}$ is the total amount of available computation resource in the cloud, and $r_i(\mathcal{G}_{i,j})$ is the amount of cloud computation resource demanded by the camera i under $\mathcal{G}_{i,j}$. $\mathcal{D}_t$ is the total amount of available computation resource on the edge node s_t, and $d_i(\mathcal{G}_{i,j})$ is the amount of edge computation resource demanded by the camera i under $\mathcal{G}_{i,j}$. $r_i(\mathcal{G}_{i,j})$ and $d_i(\mathcal{G}_{i,j})$ are given by the profiling results for the detection model.

In terms of the networking resource, we should have

$$\forall t: \sum_{\{i|s(i)=t\}} \sum_{j}^{\mathcal{N}_i} b_i(\mathcal{G}_{i,j}) \cdot x_{i,j} \quad \leq \quad \mathcal{B}_t, \quad (7)$$

Here, $\mathcal{B}_t$ be the bottleneck bandwidth of the path from s_t to the cloud, and $b_i(\mathcal{G}_{i,j})$ is the amount of networking resource demanded by the camera i, given by the profiling results for the detection model.

The problem formulated in this subsection (from Equation 2 to Equation 7) can be mapped to the *multiple-choice multi-dimensional knapsack problem* (MMKP). In [46], the authors proposed an approximation algorithm with a complexity of $O((\sum_{i=1}^{n} \mathcal{N}_i)^{2m+1})$. Obviously, the algorithm cannot be used to solve this problem due to the large value of n, $\mathcal{N}_i$, $(2m+1)$. The scheduling decision must be made in a nearly real-time manner when the system status changes, so a heuristic algorithm must be designed to solve the problem quickly.

C. Heuristic Algorithm

1) Reducing the Number of Candidate Scheduling Plans: A scheduling plan $\mathcal{G}_{i,j}$ is said to be *Pareto Optimal* [47] if there exists no other scheduling plan $\mathcal{G}_{i,j'}$ that can achieve better utility without increasing any kind of resource. Obviously, we are only interested in Pareto optimal plans because other scheduling plans would never be selected as the best solution for the camera i. Therefore, we can remove the plans that are not Pareto optimal, which can help us reduce the search space of the optimization problem. Mathematically, each $\mathcal{G}_{i,j'}$ is removed if it satisfies the following condition,

$$\exists j, \quad \text{such that} \quad \mathcal{U}_{i,j} > \mathcal{U}_{i,j'}, \quad r_i(\mathcal{G}_{i,j}) \leq r_i(\mathcal{G}_{i,j'}), \\ d_i(\mathcal{G}_{i,j}) \leq d_i(\mathcal{G}_{i,j'}), \quad b_i(\mathcal{G}_{i,j}) \leq b_i(\mathcal{G}_{i,j'}).$$

We do not define new notations for the set of scheduling plans after filtering. From now on, $\mathcal{G}_i$ denotes the set of Pareto optimal scheduling plans of the camera i instead of all possible scheduling plans.

2) Heuristics to Evaluate Plans: After Pareto optimal filtering, we need to find the optimal scheduling solution. Generally speaking, a heuristic algorithm searches for the optimal solution by replacing current solution with a better solution iteratively until no better solution can be found. Given a particular search point (*i.e.* a temporary scheduling decision), we can try to find the current best scheduling plan $\mathcal{G}_{i,j^*}$ for each individual camera stream i. Among all camera streams, we can select one stream (say i^*) and update its scheduling plan to its current best scheduling plan $\mathcal{G}_{i^*,j^*}$, which results in a new search point. In this way, we update the search point iteratively and explore the search space to find the optimal solution.

But how to find $\mathcal{G}_{i,j^*}$ for each i and how to determine i^*? Naturally, a good scheduling plan should be able to achieve a larger utility while consuming less resource. Furthermore, among multiple kinds of resource, it should be more conservative in using the resource that tends to be in short supply.

Algorithm 1: Deriving the optimal scheduling decision

Input: $\mathcal{G}_i(\forall i)$; $\mathcal{R}$; $\mathcal{D}_t(\forall t)$; $\mathcal{B}_t(\forall t)$; ϵ
Output: optimal solution $\mathcal{S}$ and its total utility $\mathcal{U}$

```
S ← initialize a feasible solution;
U, r, d_t(∀t), b_t(∀t) ← compute the utility and resource usages of S;
θ = 0;
while θ < ε and ∃i : G_i ≠ ∅ do
    for i do
        for j do
            compute γ(G_{i,j});
    let G_{i*,j*} be the scheduling plan with maximum γ ;
    remove G_{i*,j*} from G_{i*};
    S* ← update S by using G_{i*,j*};
    U*, r*, d_t*(∀t), b_t*(∀t) ← compute the current utility and resource usage of S*;
    if S* is feasible and U* > U then
        S, U, r, d_t(∀t), b_t(∀t) ← S*, U*, r*, d_t*(∀t), b_t*(∀t);
        θ = 0;
    else
        θ = θ + 1;
return S, U;
```

From this insight, we define the efficiency of $\mathcal{G}_{i,j}$ on using resources to evaluate the optimality of a scheduling plan.

$$\gamma(\mathcal{G}_{i,j}) = \frac{\mathcal{U}_{i,j}}{\rho^r \cdot r_i(\mathcal{G}_{i,j}) + \rho_t^d \cdot d_i(\mathcal{G}_{i,j}) + \rho_t^b \cdot b_i(\mathcal{G}_{i,j})} \quad (8)$$

Here, t is the edge node to which the camera i is connected, *i.e.*, $t = s(i)$. The denominator is the sum of the consumption of each resource (r_i, d_i and b_i) weighted by the shortage degree of the resource (ρ^r, ρ_t^d and ρ_t^b). ρ^r is for the cloud computation resource, ρ_t^d is for the computation resource of the edge node t and ρ_t^b is for the bandwidth resource from the edge node t to the cloud.

The shortage degree can be evaluated by the utilization rate of the resource. Taking ρ_t^b as an example, ρ_t^b can be calculated as follows,

$$\rho_t^b = \frac{\sum_{\{i|s(i)=t\}} b_i(\mathcal{G}_{i,j(i)})}{\mathcal{B}_t}.$$

Here, $j(i)$ denotes the scheduling plan index selected for the camera i currently. ρ_t^b is the result of the scheduling plans of all cameras that are connected to edge node t in the system, which means the current best scheduling plan for a camera i is related to other camera's current decisions.

Our algorithm to find the optimal scheduling decision is summarized and described in Algorithm 1. Here, ϵ is a parameter set by administrators and the algorithm is terminated if the total utility does not increase in the recent ϵ iterations. In our experiments, we set ϵ to be 10. Initially, we set the

scheduling plan of each camera stream to be the scheduling plan with minimum utility. This serves as the start point, which is denoted by $\mathcal{S}$, and we compute its resource usage and total utility (lines 1-2). θ is the number of consecutive iterations that the total utility does not increase (line 3). Based on Equation 8, we can compute $\gamma(\mathcal{G}_{i,j})$ for each scheduling plan (lines 5-7). Then, the scheduling plan with the highest efficiency, *i.e.* $\mathcal{G}_{i^*,j^*}$, is selected to be tried in the next iteration, and it is removed from the set of candidate choices. (lines 8-9). If $\mathcal{G}_{i^*,j^*}$ is feasible and it improves the total utility, we update the current solution, and continue the iterations (lines 10-16). It terminates when there is no candidate scheduling plan to explore or there is no utility improvement in recent ϵ iterations

Let $\mathcal{M}_i$ denote the number of pareto optimal scheduling plans of the camera stream i. In each iteration, there are n cameras, and at most $\sum_{i=1}^{n} \mathcal{M}_i$ scheduling plans are explored. There are at most $\sum_{i=1}^{n} \mathcal{M}_i$ iterations because each iteration removes one scheduling plan from the candidate set which consists of $\sum_{i=1}^{n} \mathcal{M}_i$ scheduling plans. Therefore, the time complexity of Algorithm 1 is $O((\sum_{i=1}^{n} \mathcal{M}_i)^2)$. In practice, Algorithm 1 terminates much earlier than $\sum_{i=1}^{n} \mathcal{M}_i$ iterations, because the resource constraints are hit, or the total utility does not increase in recent ϵ iterations. We evaluate the running time of our algorithm in Section VII-F.

VI. Non-destructive Migration Scheme

The wide-area network bandwidth between an edge node and the cloud is highly variable, and users may submit or terminate tasks during the running of the video analytics system. The system needs to keep tracking the status of resource supply and request demand, and re-evaluate its scheduling decision once a status change is detected. It is likely that a task needs to be migrated to the cloud from the edge node (or vice versa) in the new scheduling decision.

How to migrate a task is an issue that must be handled in any adaptive scheduling system. However, we have not seen any research work on the migration issues for live video analytics systems. A scheduling system that just naively stops the task in the source node (hereafter called source task) and then starts the task in the destination node (hereafter called destination task) can cause the following problems.

1) The frames arriving during a migration cannot be processed, which affects the accuracy of the tasks.
2) Some video analytics tasks need to tracking the status changes of objects, *e.g.*, detecting whether a pedestrian is running through a red light. In order to analyze a frame correctly, a task of this kind has to analyze δ frames before this frame and get the feedback information of these frames from DNN models, *e.g.*, the intermediate features and the boxes of objects in previous frames. Therefore, after the destination task is started, the analytics of initial δ frames is incorrect because the task has no feedback information of the frames before them.
3) If the task is stateful, *e.g.*, counting the number of a specific type of objects, it may lose its states.

To ensure the migrated task's accuracy, we must transmit the data required by the destination task from the source task and process the frames arriving during the migration. However, transmitting a large amount of data can induce excessive downtime, which seriously affects the latency of the migrated task. To avoid the performance degradation of migrated tasks (accuracy and latency) , we design a non-destructive migration scheme in our system. Our basic idea is to *warm up for the migrated tasks in the destination node as much as possible and make sure only necessary data is transmitted from the source node*.

A. Warm up for the migrated task

The first step in the warm-up phase is loading the container image and initialize the runtime library such as TensorFlow at the destination node. We name this step as *container startup*.

The second step in the warm-up phase is starting the migrated task in the destination node. The migrated task can have multiple components. The components that can be shared with a task in the destination node have been running in the destination node. We need to start other components and hook these new components and existing components together according to the new specification of the pipeline. This step is named as *components warm-up*.

Now we start receiving the video stream and analyzing the frames of the stream. The destination task starts to get feedback information from DNN models for δ frames. The value of δ depends on the object tracking method used by this task. In our implementation, we use deep SORT [48] as the object tracking method, and the value of δ is set to be 15 according to [48], [49]. Again, if the DNN model instance of the destination pipeline has been running, the system has had the stream and the feedback information of frames, and we do not need this step. This step is named as *frames warm-up*.

During the above three steps, the migrated task is still running normally in the source node, and the destination node is getting ready for this task.

Unfortunately, the statistic results (*e.g.*, the number of vehicles in this hour) cannot be warmed up, and it has to be transmitted to the destination node from the source node, which causes an unavoidable downtime period.

B. Procedure of non-destructive migration

Figure 8 shows the procedure of our non-destructive migration scheme.

The migration agent in the source node, *i.e.*, source agent, sends a *migration request* to the destination node (①). When the destination agent receives the request, it starts the step *container startup* and *components warm-up* (②).

Let n be the ID of the first frame received by the destination task after the components warm-up step is accomplished successfully. For simplicity, we assume that both the source and destination can receive frames from cameras at the same time. The destination sends a signal with $n + \delta$ to the source (③), indicating that the destination is capable to analyzing the frame $n+\delta$ correctly. Meanwhile, the destination task starts the

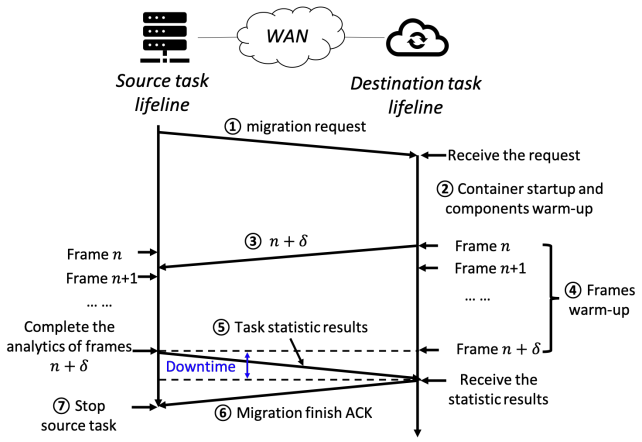

Fig. 8. An example of a task migration.

procedure of *frames warm-up* to receive and store the feedback information of frames (④) .

The source receives this signal. It keeps analyzing frames normally until the frame $n+\delta-1$. After the frame $n+\delta-1$ is analyzed, the source stops analyzing frames and sends its states information (statistic results) to the destination (⑤).

After the destination receives the state information, the destination agent pushes the state information into the migrated task and sends a *migration finish ACK* to the source (⑥). The destination task starts to be responsible for analyzing the frames after $n+\delta-1$ to accomplish the requirement of the migrated task.

The source stops the source task until receiving the *migration finish ACK* (⑦). We can see that the migration downtime is the duration to send the state information from the source to the destination.

VII. Experiment Results

The DNN models and tasks must be profiled on the same GPU type as the edge node and the cloud of the running system. Our offline profiling of *Faster R-CNN + ResNet50* (small model on edge nodes) and *Faster R-CNN + ResNet101* (big model partitioned between edge nodes and the cloud) is conducted on NVIDIA RTX 2070 Super (edge node) and NVIDIA RTX 3090 (cloud). The models are pre-trained on the Kitti dataset [50]. We select 5 levels of frame rate, 5 levels of resolution, and 5 partition schemes as candidates.

We only consider tasks relevant to the detection of two object types, *i.e.*, car and pedestrian. A camera stream can be associated with multiple tasks, and each task values accuracy and latency differently and thus it has its own utility function of the achieved accuracy and latency.

A. Experiment Settings

Experiment setup: Each edge node has 40 cores of the 2.2GHz Intel Xeon processor, 32 GB RAM, and a RTX 2070 Super GPU. The cloud has 64 cores of the 2.2GHz AMD EPYC processor, 64 GB RAM, and a RTX 3090 GPU. There are 2 edge nodes and a single cloud. Each edge node is responsible for 4 cameras, and all edge nodes are connected to the only cloud, and they are not changed across experiments. The bandwidth between an edge node and the cloud is varied between 25Mbps and 100Mbps to simulate the dynamic networking environment according to the experimental measurements in [6]. We use Cgroup to control the bandwidth between edge node and cloud.

Dataset: We select 8 videos from YouTube as the input camera streams using the keyword "highway traffic video HD". The length of each video stream is one hour, and we loop the video if the original video length is less than one hour. The quality of each video is (1080p, 30fps). These videos do not have human-annotated ground truth, and we have to use the estimated accuracy from the offline profiling results. Other key metrics (*e.g.*, latency, GPU usage, and the number of edge or cloud pipelines) are collected during our real-world experiments.

Utility function: The utility function of each task is specified by its user when the user submits the task. We assume that the user's utility function is defined as follows.

$$u(a,l) = f_{\rho_a,\tau_a}(a) - f_{\rho_l,\tau_l}(l)$$

$$f_{\rho,\tau}(x) = \begin{cases} 0 & \text{if} \quad x \leq \tau \\ \rho(x-\tau) & \text{if} \quad x > \tau \end{cases}$$

Basically, the performance metrics of an task, *i.e*, accuracy and latency, affect the utility only when the corresponding performance metrics exceeds a specified threshold τ. ρ is used to indicate the significance of the corresponding metrics, *i.e.*, accuracy (ρ_a) and latency (ρ_l).

Each task has its own parameters ρ_a, ρ_l, τ_a and τ_l. In this way, we can simulate tasks of various performance considerations. In our evaluation, τ_a is between 0.4 and 0.6, τ_l is between 0.1s and 0.4s, and ρ_a and ρ_l are integers between 1 and 10. We generate tasks with random parameters within these ranges to simulate a random workload for experiments, and then we gradually change ρ_a and ρ_l to simulate more accuracy-sensitive tasks and more latency-sensitive tasks.

B. Benchmarks

We compare our solution with the following solutions.

- **Optimal**: It is the solution to the optimization problem derived using Gurobi [51], a powerful optimization solver. It can be used to evaluate the optimality of our heuristic algorithm and can be viewed as an upper bound.
- **VE**: VideoEdge [5] also studies a scenario with multiple cameras, multiple edge nodes and a cloud. It aims to optimize accuracy of tasks. To make two solutions comparable, we modify the optimization goal of VideoEdge to maximize utility.
- **AGP**: It is proposed in [38]. The work makes decision for each task on which model to be used among multiple candidate models. To make two solutions comparable, we profile more models (*i.e.*, Yolo and SSD + MobileNet) under different video qualities.

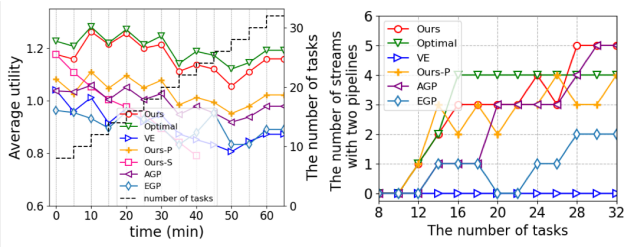

Fig. 9. The performance comparison under workloads of different intensities.

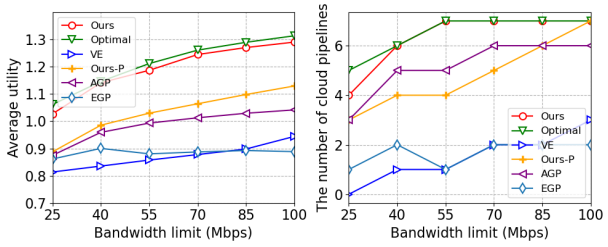

Fig. 10. The performance of different methods under different bandwidth.

- **EGP**: It is an alternative method to the problem proposed in [38].
- **Ours-P**: It is a variant of our solution, which does not allow DNN partition. It can be used to present the benefit retrieved from DNN partition with compression.
- **Ours-S**: It is a variant of our solution, which does not allow sharing of model computation.

C. Performance under Increasing Workloads

We generate 32 tasks, and each task is with a random parameter setting. During this experiment, we submit 2 tasks every 5 minutes and gradually increase the number of active tasks from 8 to 32, which is to simulate the workloads of different intensities. The networking bandwidth is set to be 50Mbps and kept fixed during the experiment.

The results are presented in Figure 9. We can see that our solution is always close to the Optimal derived by the optimization solver, and the performance gap is within 5.6%, which demonstrates the efficiency of our heuristic algorithm. Our system performs significantly better than VE, AGP and EGP. As the workload increases, the average utility of Ours-S drops sharply while the average utility of other methods are relatively stable. Since the average utility of Ours-S is always poor and Ours-S cannot find a feasible solution when the number of tasks exceeds 22, we do not include Ours-S for comparison in the experiments in later subsections. It also shows that the performance of our solution is much better than Ours-P, which demonstrates that DNN partition with compression improves the system performance. We also plot the number of camera streams with both edge pipeline and cloud pipeline. As the number of tasks increases, there tend to be more tasks on the same camera and these tasks can be with competing/diverse performance goals, therefore more streams have to be analyzed in two places for different tasks.

D. Performance under Different Bandwidths

We set the number of tasks to be 24 and keep their parameters of utility functions the same as the original values generated randomly. We increase the network bandwidth every 5 minutes. The networking bandwidth is changed from 25Mbps to 100Mbp. As shown in Figure 10, we can see that the performance of Ours is close to Optimal, and outperforms all other solutions. As the available networking resource increases, more video streams can be transmitted to the cloud for better accuracy. Therefore, the number of cloud pipelines also

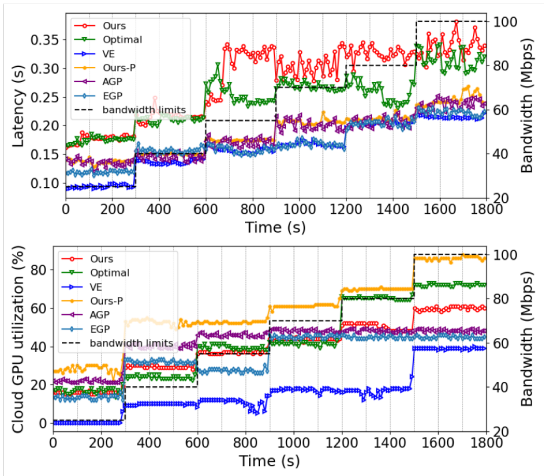

Fig. 11. The latency and cloud GPU utilization under different bandwidth.

increases. We find that the number of cloud pipelines in our system and Optimal is always more than all other solutions. This shows that DNN partition can make more videos to be streamed to the cloud.

Figure 11 shows the latency and the cloud GPU utilization over time. The system runs for 30 minutes, and we calculate the average latency and GPU utilization every 10s. The latency of the tasks increases with the increase of network bandwidth, because there are more cloud pipelines in the cloud. The average latency of Ours and Optimal is larger than other solutions. The reason is that Ours and Optimal place more accuracy-sensitive pipelines in the cloud than other methods. It improves the utilities of the associated tasks while the latencies of these tasks increase.

E. Performance under More Accuracy-Sensitive Workloads

The bandwidth is set to be 50Mbps. We still use the same set of tasks, and the number of tasks is 24. During this experiment, ρ_a of all tasks is increased from 1 to 10 while other parameters remain unchanged. The duration of the experiment is 50 minutes.

Figure 12 shows the results. The utility achieved by our solution is very close to Optimal, and the gap between them is within 1%. Both our solution and Ours-P are better than other solutions. As ρ_a increases, the number of cloud pipelines is increasing. It indicates that more tasks are placed in the cloud because accuracy becomes more important for these tasks,

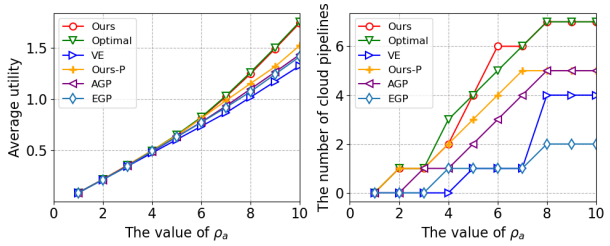

Fig. 12. The performance as the tasks are more and more accuracy-sensitive.

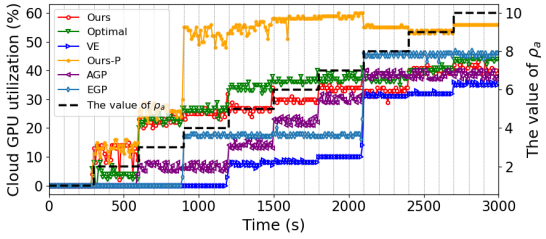

Fig. 13. The cloud GPU utilization as the tasks are more and more accuracy-sensitive.

and the cloud pipelines can achieve higher accuracy. DNN partition can enable more cloud pipelines as it helps to reduce the networking consumption of a single pipeline. That can also explain why we see more cloud pipelines in our system than all other solutions in the second subplot of Figure 12.

Figure 13 shows the cloud GPU utilization. We can see the cloud GPU utilization is increasing as ρ_a increases. When the value of ρ_a is big, the cloud GPU utilization under Ours-P is higher than Ours and Optimal but the number of cloud pipelines under Ours-P is less than Ours and Optimal. The reason is that the DNN partition pushes some layers of big model to the corresponding edge nodes, alleviating the workload in the cloud.

We also conduct experiments to evaluate the performance under more latency-sensitive workloads which presents similar observations. Due to page limitation, we show the experiment results in the supplemental materials.

F. Performance under Massive Video Streams

We conduct a simulation in a scenario with more video streams. There are 50 edge nodes and a cloud. Each edge node has two RTX 2070 Super GPUs and is responsible for 10 cameras. The cloud has eighty RTX 3090 GPUs. The number of video analytics tasks varies from 500 to 2500 during experiments. The bandwidth between an edge node and a cloud is 100Mbps. We evaluate the performance of different methods under workloads of different intensities.

Figure 14 shows the results. The utility achieved by our solution is close to the Optimal, and the gap between them is within 5.1%. The average utility of our system is better than other methods. Besides, we also show the time for our scheduler and Optimal to compute the scheduling solution. As the number of tasks increases, the time needed by both methods is increasing, but the time of our scheduler is significantly

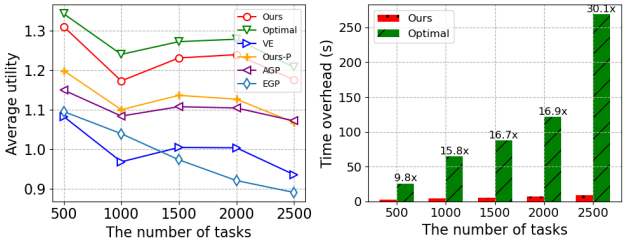

Fig. 14. The performance of different methods under massive video streams.

smaller than Optimal. When the number of tasks is 2500, our global scheduler can compute the optimal scheduling solutions in 10 seconds, which is acceptable for our video analytics system.

G. Evaluation of Non-destructive Migration Scheme

1) Time cost of warm-up phase: Our migration scheme needs to warm up for the migrated tasks in the destination node, and the time cost of the warm-up phase directly affects the total migration time. Therefore, we conduct measurements for the time cost of warm-up phase during migration. We conduct experiments for two popular tasks, *i.e.*, vehicle counter and pedestrian recognition. The task is migrated from an edge node to the cloud. In order to show the worst-case performance, we assume there is no component that can be shared in the destination node, so all components of the task have to be loaded and initialized during warm-up phase. Intuitively, the destination node needs to receive δ frames to complete frame warm-up, so δ and frame rate would affect the time cost of frame warm-up. In this experiment, δ is set to be 10 and 15, which has been large enough for the accuracy of object tracking. The resolution is set to be 720p, and the frame rate is set to be 5, 10, or 15.

We conduct experiments for each setting five times and show the results in Table I. Since the results of pedestrian recognition are similar as vehicle recognition, we only show the results of vehicle counter task. From the table, we can see that the time costs of container startup and component warm-up are independent of frame rate and δ, and they take about 10 seconds. The frame warm-up takes more time as δ increases or frame rate decreases. The maximum average time cost of these three steps is 11.96 seconds and the minimum is 9.70 seconds. Since the migrated task keeps running on the source node during the warm-up phase, the time overhead of the warm-up phase is acceptable.

2) Performance of migration scheme: We compare our migration scheme with "stop-and-start" migration. We implement a simple "stop-and-start" migration, in which we stop the migrated task at the source node, start a copy of migrated task on the destination, and then transmit the video stream, states and unprocessed frames during task downtime to the destination. There is no warm-up phase in "stop-and-start" migration. We conduct an experiment to migrate a vehicle counter task from an edge to the cloud. The network

TABLE I
TIME COST OF WARM-UP PHASE.

(fps, δ)	Container startup (s)		Component warm-up (s)		Frame warm-up (s)		Total (s)	
	Ave.	Wor.	Ave.	Wor.	Ave.	Wor.	Ave.	Wor.
(5, 15)	1.79	1.88	6.97	7.62	3.20	3.22	11.96	12.68
(10, 15)	1.85	1.86	7.46	7.66	1.62	1.66	10.93	11.16
(15, 15)	1.81	1.87	7.77	7.94	1.35	1.36	10.94	11.15
(5, 10)	1.75	1.86	7.15	7.89	2.28	2.29	11.17	12.01
(10, 10)	1.81	1.89	7.27	7.76	1.22	1.23	10.30	10.83
(15, 10)	1.79	1.86	6.98	7.69	0.92	0.96	9.70	10.45

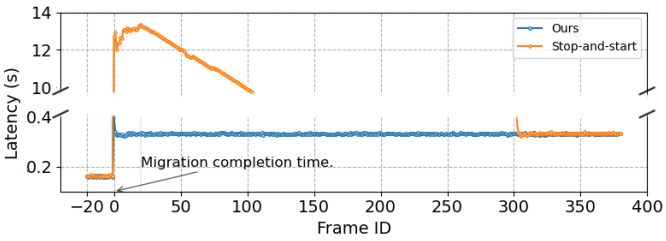

Fig. 15. The latency of each frame of the migrated task.

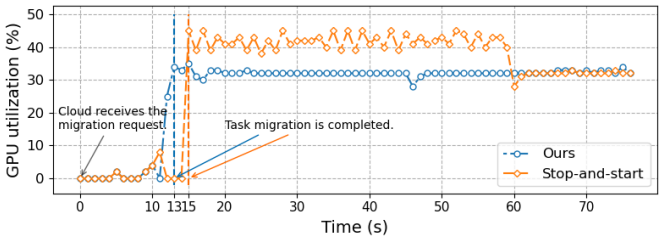

Fig. 16. The GPU utilization of cloud at different migration schemes.

bandwidth is set to be 50 Mbps, and the video quality is (720p, 5fps).

Figure 15 shows the latency of each frame before and after a task migration. The first frame analyzed by the destination task is denoted as frame 0. In our migration scheme, the latency of frame 0 is larger than the frames after it, and the gap between the latency of frame 0 and the average latency of later frames is 59ms. The longer latency of frame 0 is caused by the necessary transmission of statistic results from the source to the destination, *i.e.*, migration downtime. But in "stop-and-start" migration, the gap is 12.39s, and the latency of more than 300 frames is larger than the average latency of later frames. It shows that "stop-and-start" migration causes long downtime and the performance of the task degrades for a longer time.

Figure 16 shows the GPU utilization during the migration. After the task migration is completed, the GPU utilization of our migration scheme is consistently around 32%, while the GPU utilization of "stop-and-start" migration is significantly higher than 32% at the first tens of seconds. This is because the cloud needs to process a lot of queued frames under "stop-and-start" migration (in this experiment, 72 frames are queued), while only 1 queued frame needs to be processed with our warm-up phase in our migration scheme.

In summary, our migration scheme enables a near-seamless migration without perceptible impact on system computation load and task latency, while "stop-and-start" migration causes performance degradation for a long period.

VIII. CONCLUSION

In this paper, we consider the scenario in which each camera stream can be used by multiple tasks for their own purposes, and propose an adaptive system to optimize the performance of all these video analytics tasks, *i.e.*, accuracy and latency, in a camera-edge-cloud architecture. The system enables sharing of model computation as much as possible, in which a stream can have two pipelines, one for a small model on an edge node, and one for a big model in the cloud. Our system also exploits DNN partition technique to trade a small amount of edge computation resource for scare wide-area networking resource when necessary. By profiling *Faster R-CNN + ResNet101*, we report that compressing feature maps is necessary to make the partition of this complex model useful. We propose a heuristic algorithm to maximize the total utility of all tasks quickly. Furthermore, sometimes tasks might be required to migrate because the scheduling decision made by the system changes to be adaptive to the changing resource supply and demand. In order to avoid the performance degradation during migration, we comprehensively analyze the possible reasons of performance degradation during migration, and design a non-destructive migration scheme to ensure the performance of tasks during migration. Simulations demonstrate our analytics system performs better than state-of-the-art solutions.

REFERENCES

[1] IHS, "Security technologies: Top trends for 2020 ihs," https://cdn.ihs.com/www/pdf/1218/IHSMarkit-Security-Technologies-Trends-2019.pdf.

[2] Avigilon, "Products : Know what is happening so you can act with certainty," http://avigilon.com/products/.

[3] Microsoft, "The future of computing: intelligent cloud and intelligent edge," https://www.microsoft.com/en-us/internet-of-things/intelligentedge.

[4] J. Ren, Y. Guo, D. Zhang, Q. Liu, and Y. Zhang, "Distributed and efficient object detection in edge computing: Challenges and solutions," *IEEE Network*, vol. 32, no. 6, pp. 137–143, 2018.

[5] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 115–131.

[6] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, "Awstream: Adaptive wide-area streaming analytics," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 236–252.

[7] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1421–1429.

[8] J. Wang, Z. Feng, Z. Chen, S. George, M. Bala, P. Pillai, S.-W. Yang, and M. Satyanarayanan, "Bandwidth-efficient live video analytics for drones via edge computing," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 159–173.

[9] P. Yang, F. Lyu, W. Wu, N. Zhang, L. Yu, and X. S. Shen, "Edge coordinated query configuration for low-latency and accurate video analytics," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 7, pp. 4855–4864, 2019.

[10] Z. Lu, K. S. Chan, and T. La Porta, "A computing platform for video crowdprocessing using deep learning," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1430–1438.

[11] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 377–392.

[12] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: scalable adaptation of video analytics," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 253–266.

[13] K. Hsieh, G. Ananthanarayanan, P. Bodik, S. Venkataraman, P. Bahl, M. Philipose, P. B. Gibbons, and O. Mutlu, "Focus: Querying large video datasets with low latency and low cost," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 269–286.

[14] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "Noscope: optimizing neural network queries over video at scale," *arXiv preprint arXiv:1703.02529*, 2017.

[15] C. Canel, T. Kim, G. Zhou, C. Li, H. Lim, D. G. Andersen, M. Kaminsky, and S. R. Dulloor, "Scaling video analytics on constrained edge nodes," *arXiv preprint arXiv:1905.13536*, 2019.

[16] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *Proc. IEEE INFOCOM*, 2020, pp. 1–10.

[17] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[18] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[19] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive dnn surgery for inference acceleration on the edge," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1423–1431.

[20] "Large-scale video analytics," https://github.com/lolo-pop/scalable-video-analytics.

[21] Y. Huang, F. Wang, F. Wang, and J. Liu, "Deepar: A hybrid device-edge-cloud execution framework for mobile deep learning applications," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2019, pp. 892–897.

[22] T. Mohammed, C. Joe-Wong, R. Babbar, and M. Di Francesco, "Distributed inference acceleration with adaptive dnn partitioning and offloading," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 854–863.

[23] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[24] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–16.

[25] T. Tan and G. Cao, "Fastva: Deep learning video analytics through edge processing and npu in mobile," *arXiv preprint arXiv:2001.04049*, 2020.

[26] Y. Wang, W. Wang, J. Zhang, J. Jiang, and K. Chen, "Bridging the edge-cloud barrier for real-time advanced vision analytics," in *11th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, 2019.

[27] Y. Wang, W. Wang, D. Liu, X. Jin, J. Jiang, and K. Chen, "Enabling edge-cloud video analytics for robotics applications," in *Proceedings of the IEEE International Conference on Computer Communications, Virtual Conference*, 2021, pp. 10–13.

[28] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, 2015, pp. 155–168.

[29] K. Chen, T. Li, H.-S. Kim, D. E. Culler, and R. H. Katz, "Marvel: Enabling mobile augmented reality with low energy and low latency," in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, 2018, pp. 292–304.

[30] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. Netravali, "Reducto: On-camera filtering for resource-efficient real-time video analytics," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 359–376.

[31] C. Long, Y. Cao, T. Jiang, and Q. Zhang, "Edge computing framework for cooperative video processing in multimedia iot systems," *IEEE Transactions on Multimedia*, vol. 20, no. 5, pp. 1126–1139, 2017.

[32] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang, "Server-driven video streaming for deep learning inference," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 557–570.

[33] W. Zhang, S. Li, L. Liu, Z. Jia, Y. Zhang, and D. Raychaudhuri, "Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1270–1278.

[34] X. Zeng, B. Fang, H. Shen, and M. Zhang, "Distream: scaling live video analytics with workload-adaptive distributed edge intelligence," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 409–421.

[35] M. Ali, A. Anjum, O. Rana, A. R. Zamani, D. Balouek-Thomert, and M. Parashar, "Res: Real-time video stream analytics using edge enhanced clouds," *IEEE Transactions on Cloud Computing*, 2020.

[36] S. Wang, S. Yang, and C. Zhao, "Surveiledge: Real-time video query based on collaborative cloud-edge deep learning," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2519–2528.

[37] Y. Zhang, J.-H. Liu, C.-Y. Wang, and H.-Y. Wei, "Decomposable intelligence on cloud-edge iot framework for live video analytics," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8860–8873, 2020.

[38] N. Hudson, H. Khamfroush, and D. E. Lucani, "Qos-aware placement of deep learning services on the edge with multiple service implementations," *arXiv preprint arXiv:2104.15094*, 2021.

[39] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, 2005, pp. 273–286.

[40] M. R. Hines, U. Deshpande, and K. Gopalan, "Post-copy live migration of virtual machines," *ACM SIGOPS operating systems review*, vol. 43, no. 3, pp. 14–26, 2009.

[41] L. Hu, J. Zhao, G. Xu, Y. Ding, and J. Chu, "Hmdc: Live virtual machine migration based on hybrid memory copy and delta compression," *Appl. Math*, vol. 7, no. 2L, pp. 639–646, 2013.

[42] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[43] P. G. Lindstrom *et al.*, "Fpzip," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2017.

[44] P. Ratanaworabhan, J. Ke, and M. Burtscher, "Fast lossless compression of scientific floating-point data," in *Data Compression Conference (DCC'06)*. IEEE, 2006, pp. 133–142.

[45] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[46] B. Patt-Shamir and D. Rawitz, "Vector bin packing with multiple-choice," *Discrete Applied Mathematics*, vol. 160, no. 10-11, pp. 1591–1600, 2012.

[47] Y. Censor, "Pareto optimality in multiobjective problems," *Applied Mathematics and Optimization*, vol. 4, no. 1, pp. 41–59, 1977.

[48] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *2017 IEEE international conference on image processing (ICIP)*. IEEE, 2017, pp. 3645–3649.

[49] W. Liu, G. Kang, P.-Y. Huang, X. Chang, Y. Qian, J. Liang, L. Gui, J. Wen, and P. Chen, "Argus: Efficient activity detection system for extended video analysis," in *Proceedings of the IEEE Winter Conference on Applications of Computer Vision Workshops*, 2020, pp. 126–133.

[50] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[51] Gurobi, https://www.gurobi.com.