RES: Real-time Video Stream Analytics using Edge Enhanced Clouds

M Ali, A Anjum, O Rana, A.R Zamani, D Balouek-Thomert, M Parashar

Abstract—With increasing availability and use of Internet of Things (IoT) devices such as sensors and video cameras, large amounts of streaming data is now being produced at high velocity. Applications which require low latency response such as video surveillance, augmented reality and autonomous vehicles demand a swift and efficient analysis of this data. Existing approaches employ cloud infrastructure to store and perform machine learning based analytics on this data. This centralized approach has limited ability to support real-time analysis of large-scale streaming data due to network bandwidth and latency constraints between data source and cloud. We propose RealEdgeStream (RES) an edge enhanced stream analytics system for large-scale, high performance data analytics. The proposed approach investigates the problem of video stream analytics by proposing (i) filtration and (ii) identification phases. The filtration phase reduces the amount of data by filtering low-value stream objects using configurable rules. The identification phase uses deep learning inference to perform analytics on the streams of interest. The phases consist of stages which are mapped onto available in-transit and cloud resources using a placement algorithm to satisfy the Quality of Service (QoS) constraints identified by a user. We demonstrate that for a 10K element data streams, with a frame rate of 15-100 per second, the job completion in the proposed system takes 49% less time and saves 99% bandwidth compared to a centralized cloud-only based approach.

Index Terms—IoT, edge computing, video stream analytics, real-time analytics, deep learning, software defined networks, big data, Cloud Computing

1 Introduction

VIDEO cameras are the most versatile form of IoT devices. The number of video cameras have grown at an unprecedented rate to increase public safety and security around the globe. France and UK have ≈1 million and ≈6 million CCTV cameras installed respectively. They are currently being monitored by human personnel who incessantly stare at a grid view screen to monitor actions or events of interest. Video surveillance has traditionally been performed by operators watching one or more video feeds, the limitations of which are well known. About 90% of incidents are missed after 20 minutes [1] as the concentration of the CCTV operators drops.

Video analytics relieves the operator by automating the surveillance process and can proactively detect, recognize and respond to events coming from video streams. However, performance and accuracy are the two major concerns in video analytics. The accuracy of the analytics process has been improving by leaps and bounds by novel approaches in deep learning models over recent years. Analytics using deep learning demand powerful compute and storage resources which are offered by many existing cloud platforms. Therefore, most existing video stream analytics systems [2] [3] [4] employ cloud infrastructure to perform operations such as object classification and recognition on the incoming data streams. However, performance is a major concern for cloud-based stream analytics systems due to network latency and limited bandwidth.

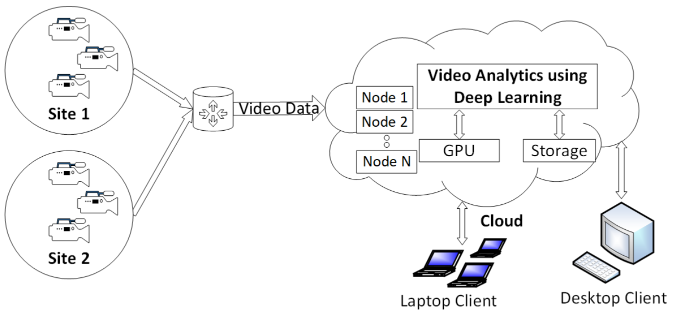


Fig. 1. Traditional Centralized Video Stream Analytics

As shown in Fig. 1, in a "traditional" cloud-based video stream processing system, video cameras are attached to a network which forwards video data to a cloud platform. The software running on the cloud performs basic processing such as video loading and motion detection in video frames. After basic processing, video frames are passed on to a machine learning model to support object classification and recognition. In such cloud-based video processing systems, all video streams are first transferred to the cloud platform where they are stored and analysed – sometimes referred to as *store first, analyze later* approach.

With significant increase in the number of IoT devices and the availability of cameras with high definition (HD), 4K and 360 videos, the load on the cloud has increased sub-

- *M.Ali is with the College of Engineering and Technology, , University of Derby, UK. Email: m.ali@derby.ac.uk.*
- *A.Anjum is with the Department of Computer Science, University of Leicester, UK. Email: a.anjum@leicester.ac.uk*
- *O.Rana is with the Department of Computer Science and Informatics, Cardiff University, UK. Email: ranaof@cardiff.ac.uk*
- *A.R Zamani, D Balouek-Thomert and M.Parashar are with the Rutgers Discovery Informatics Institute (RDI^2), Rutgers University, New Brunswick, New Jersey, US. Emails: alireza.zamani@rutgers.edu, daniel.balouek@rutgers.edu, parashar@rutgers.edu*

Manuscript received May 20, 2019; revised Mar 11, 2020.

stantially while the network bandwidth has not kept pace. Therefore, a need arises to analyze this data in real-time for applications which require low-latency response such as video surveillance, autonomous vehicles, robotics and augmented reality. However, cloud-based stream processing suffers from latency issues as it takes time to transport the data to the cloud. It is not feasible to provide low-latency and real-time analytics using a cloud-only system, due to large bandwidth requirements for transferring data from source to the cloud. Edge and Fog computing [5] complement the cloud platform to provide a viable distributed mechanism to serve low-latency requirements of many applications. We propose an edge computing based video stream analytics approach to overcome these challenges, with a focus on distributing the video processing pipeline among available resources from edge to the cloud.

The proposed approach decomposes the stream processing pipeline into two phases: a filtration phase and an identification phase. The phases consist of stages that are distributed over in-transit and cloud resources. The filtration phase filters incoming data streams and the identification phase performs analysis on the data streams. We will be using object recognition as a scenario for real-time video processing. The main contributions of this paper are given below.

- A novel video analytics pipeline is proposed that consists of filtration and identification phases, with three stage types: basic, filter and machine learning. The filtration phase allows the early discarding of low-value data to improve performance. The stages are then distributed across the edge, in-transit and cloud resources, based on the resource requirements of each stage. More computationally intensive stages may continue to execute on cloud resources, whereas basic and filter stages can be executed closer to the user on edge and in-transit resources.
- A horizontal-vertical (HV) scalable architecture is designed and implemented, which can efficiently scale with an increase in input data size. This scalability is achieved by utilizing hardware resources, including multiple edge, in-transit and cloud nodes, to continuously provide high-performance stream analytics.
- A placement algorithm is proposed and implemented to optimally map the pipeline stages onto computational resources that are part of the HV architecture, to achieve real-time Quality of Service (QoS) performance requirements.
- A configurable rule-based analytics approach is proposed for filtering video streams based on the input criteria, such as the detected object type and the inference confidence threshold value.

The rest of this paper is organised as follows. In section 2 we briefly discuss the state of the art in large-scale, high-performance stream analytics systems. In section 3, the research approach adopted in this work is outlined using a two phase filtration-identification approach to accelerate video processing. In section 4, a generic Horizontal-Vertical (HV) architecture for stream analytics is proposed which can scale to a large number of available resources. Section 5 presents a video analysis use case to show the efficacy of the proposed approach as compared to a centralized cloud-only approach. Sections 6 and 7 describe the experimental setup and results respectively. Finally, section 8 concludes the paper demonstrating that edge-enhanced video stream processing is more efficient than a central cloud-only approach.

2 Related Work

In this section we discuss related work focusing on cloud-based stream processing. State of the art in edge enhanced stream analytics is also described to better contextualise the approach presented in this paper.

2.1 Stream Processing in Clouds

Cloud computing provides scalable compute and storage resources to meet the demand of large-scale data analysis. The use of a cloud platform to support stream processing has been successfully exploited in many systems, e.g. [6] [7] employ cloud-based compute and storage resources to store and process streaming data. Processing a large amount of data became mainstream with the Hadoop [8] [9], especially with the integration of stream processing tools such as Apache Kafka [10], Apache Spark [11] and Apache Storm [12] which enable faster processing on the cloud platform. Whereas "traditional" Hadoop deployments were mainly aimed at processing batch (pre-collected) data, Apache Spark, Kafka & Storm support processing of real time data streams – focusing on the use of in-memory processing, stream sharding and dynamic node deployment capability.

Most cloud platforms employ Graphical Processing Units (GPUs) to accelerate the data analysis pipeline and reduce the time to perform compute-intensive algorithms. GPU-based approaches have been reported in [2] [13] [14], which support deep learning on a cloud platform for scalable and accurate object detection and classification. Availability of Vision Processing Units (VPUs) from Intel further enhance the processing capability of video-based data analysis. The accuracy of deep learning models can be further improved by optimization of hyper-parameters associated with these models using a cloud platform – where different parameter combinations can be validated concurrently [15]. VideoStorm [16] is a multi-query based video analysis system which can process a large number of live video queries on large computational clusters.

However, these tools and systems can either process limited sized data streams within a deadline or struggle to offer high-performance analytics when data rates become high. These platforms demand data streams to be sent to the cloud before analysis can be initiated, leading to significant bandwidth demand and processing delays.

2.2 Edge-Enhanced Clouds

Applications such as video surveillance and autonomous vehicles demand low-latency processing capabilities which cloud-only approaches are not able to provide – as the data movement to cloud incurs a significant delay. To mitigate this issue, edge computing [17] has been proposed to provide compute and storage resources near to the source of the data. Edge computing extends cloud computing for

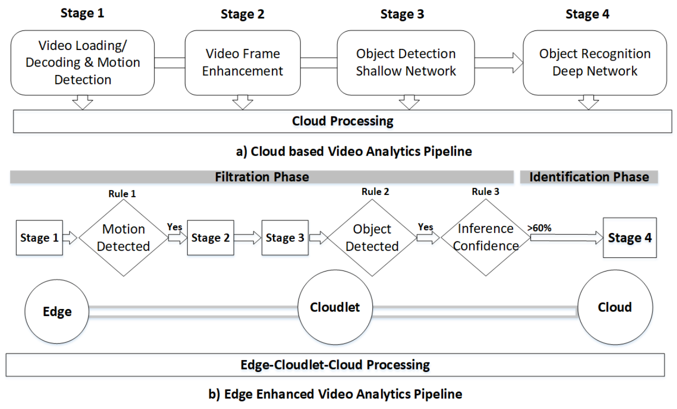


Fig. 2. Video Analysis Pipeline

latency-sensitive applications. Benefits of edge computing include a reduced requirement of storage and computational resources at the cloud, reduction of bandwidth, and provisioning of optional privacy and location services close to the edge.

Geographically distributed edge and cloud nodes are the only feasible solution to meet the real-time requirement for video surveillance. Decomposition of deep learning pipeline have been considered in [18] where edge computing was used to make the system more efficient compared to a cloud-only approach. Real-time video processing is considered as a *killer* application for edge computing [19], however scalable real-time analytics remains an important challenge – as considered in this paper.

2.3 Edge Enhanced Stream Processing Systems

Many edge-based stream analytics systems focus on deadline driven processing [20], bandwidth limited scheduling [17] [21] and real-time processing [22] [23]. Authors in [20] investigated video analytics using edge and in-transit network nodes with a deadline-time for each job and priority-based scheduling. However, their focus was on getting more jobs accepted and real-time performance, bandwidth conservation and analytics using deep learning models were not discussed.

Gigasight [17] saves bandwidth by running computer vision algorithms on a cloudlet and sending the resulting reduced data (recognized objects) to the cloud. Vigil [21] is an edge-based wireless surveillance system which saves bandwidth by scheduling techniques to support intelligent tracking and surveillance. The priority of Vigil and Gigasight is to conserve bandwidth while our priority is to optimize performance while saving as much bandwidth as possible.

Authors in [22] employ edge computing to perform basic processing by rescaling the frame with increasing algorithm complexity to achieve high-performance, however, their approach comes at the cost of accuracy. Approaches like EdgeEye [23] and R-Pulsar [24] provide API based frameworks to make it easier to deploy data and code on edge devices and are complementary to our approach.

Tetrium [25] is a wide area data processing system with multiple resources to provide geo-distributed resource allocation to tasks. In contrast to Tetrium, our data sources are located at the edge of the network and data follows a path from edge resources to a cloud. As such, our approach to stream analytics poses unique challenges which differ from Tetrium. Similar to Tetrium, VideoEdge [25] is a stream processing system which uses a hierarchical computational cluster to process video queries to improve the accuracy of results, with limited focus on performance enhancement. Additionally in VideoEdge, no emphasis is put on filtering the data, whereas discarding the data at an early stage of analysis is a core part of our approach to increase system efficiency.

In contrast to these existing systems and approaches, our focus is on achieving real-time performance while consuming minimum network bandwidth. In RealEdgeStream (RES), a particular emphasis is put on scalable analytics with the same level of accuracy as a cloud platform.

3 Research Approach

In traditional stream processing, the analysis process is executed on a central cloud without distinction of the individual parts or components that make up the stream processing pipeline. The functional partitioning of the analysis pipeline across the distributed edge to cloud continuum is an important challenge [26] to optimize execution performance.

Our approach involves decomposing video stream analysis into a filtration and an identification phase. The filtration phase discards *low value* data by detecting and passing only objects of interest to the next phase. The identification phase performs detection and analysis of objects which have passed the filtration phase. These phases are further decomposed into: i) basic, ii) filter and iii) machine learning stages. The two-phase design with three types of stages allows for convenient and logical functional partitioning of the stream analysis pipeline. For instance, object recognition can be decomposed into frame loading, frame enhancement and recognition stages. The total number of stages depends on the specific scenario being considered.

The video analysis pipeline for object recognition on the cloud is shown in Fig. 2a. In this approach, all of the stages from 1-4 are executed on a central cloud. The proposed decomposition of the pipeline using edge enhanced approach is shown in Fig.2b. It consists of two phases and four stages. The rules 1-3 are configurable rules which represent the filter stages and are not part of the analysis pipeline. The stages are mapped onto the available edge and in-transit resources using a placement algorithm defined in section 4.1 to make the video analysis process more efficient without a loss of accuracy.

The filtration phase may consist of basic, machine learning and filter stages to filter data based on a predefined criteria such as motion/object detection and object inference confidence. Normally, the filtration phase resides on the edge and in-transit resources to reduce the amount of data that has to be sent to the cloud. The identification phase only consists of machine learning stages, and can be executed at the in-transit or cloud nodes to satisfy QoS requirements. For object recognition, the identification phase consists of a deep learning model, as explained in section 5.1.

The stages above can be distributed among in-transit (edge/cloudlet) and cloud resources in many ways which

TABLE 1
Notations

Symbol	Description
P	Video Analytics Problem
F_P	Filtration Phase
I_P	Identification Phase
B	Set of basic stages
F	Set of filter stages
M	Set of machine learning stages
S_T	Total number of stages
R_T	Total number of resources
C_j	Time cost for a single job
C_T	Time cost for all jobs
G_s	Percentage efficiency gain

determines the performance of the system. In general when designing an edge enhanced stream analytics pipeline, stages should satisfy the following properties to make the analytics process accurate and reliable.

1) Serialization: Stages can have dependency constraints, and need to be executed in order – e.g. stage 2 must execute after stage 1 has been executed.
2) Integrity: The design of stages is a logical process. Stages should be designed to maintain their conceptual integrity based on their execution behaviour and characteristics. If all the executed stages have the same behaviour and characteristics as executing on a single machine, this property is satisfied.
3) Completion: A frame is completed or processed when all of its stages have been executed successfully and satisfies properties (1) and (2) above.

3.1 Stream Analysis Pipeline Model

Video stream processing (P) involved filtration (F_P) followed by identification (I_P), as outlined below:

$$P = F_P \rightarrow I_P \tag{1}$$

The arrow represents the functional dependency between the filtration and identification phases. Table 1 shows the notation used in this section. The filtration phase consists of a set of basic and filter stages whereas identification phase consists of machine learning stages given by:

$$F_P = B \cup F \tag{2}$$

$$I_P = M \tag{3}$$

B is the set of basic stages, F is the set of filter stages and M is the set of machine learning stages. The identification phase is usually the most compute-intensive, as it contains machine learning stages.

For basic processing stages, a transformation function can include image smoothing/sharpening, noise reduction, histogram equalization, image scaling or any other related function. For the machine learning stages, the transformation function is a machine learning model.

Filter stages perform filtering of data which is not likely to be of interest using a set of configurable rules and can be implemented using a rules engine such as Drools [27]. It decides whether to forward frames to the next node based on these rules, e.g. an exact match to an object type or an inference confidence for an object exceeding a threshold.

The sum of the total number of stages of all types is given by S_T

$$S_T = |B| + |F| + |M| \tag{4}$$

The symbols $|B|$, $|F|$ and $|M|$ represent the cardinalities of the basic, filtration and machine learning stages set.

The sum of the total number of resources R_T is given by

$$R_T = |E| + |I| + |C| \tag{5}$$

where $|E|$, $|I|$ and $|C|$ represent the total number of edge, in-transit and cloud resources respectively. The mapping of stages to resources is an important criterion which can affect the performance of the system. One simple but inefficient way of mapping stages to resources is to simply divide the total number of stages to resources. However, this approach suffers from several limitations, e.g. resource waste when all stages can be successfully executed on a single resource while satisfying the job deadline requirements. We propose a heuristic placement algorithm for the intelligent assignment of stages to resources based on the criteria of latency and deadline time in Algorithm 1.

Now, we derive the formula for the efficiency gain. The efficiency gain will allow us to compare the performance of the system with deployment over a cloud platform. Each frame coming from the video stream is considered as a job to be processed by the system.

The time cost to process each job with algorithm A is:

$$C_j = t_A + t_N \tag{6}$$

where t_A is the time to process the job and t_N is the time to transfer the job from source to the destination. The total cost C_T to process all jobs can be specified as:

$$C_T = \sum_{n=1}^{N} C_{j_n} \tag{7}$$

where n in C_{j_n} indicates the cost of the n^{th} job and the upper limit of summation N indicates the total number of jobs. A percentage gain is defined below to show the efficiency of a configuration (x) in terms of time and cost for a reference cloud (c) configuration.

$$G_s = (C_T(c) - C_T(x)) * 100/C_T(c) \tag{8}$$

where $C_T(x)$ is the total cost to process all jobs on configuration C and $C_T(c)$ is the cost to process all jobs on a cloud configuration. The cloud configuration indicates the traditional approach to video stream analysis with no edge or in-transit resources in-between the data source and the cloud. The percentage gain allows us to compare the traditional cloud approach with one of our configurations containing one or more edge and in-transit nodes to demonstrate benefit of the proposed system.

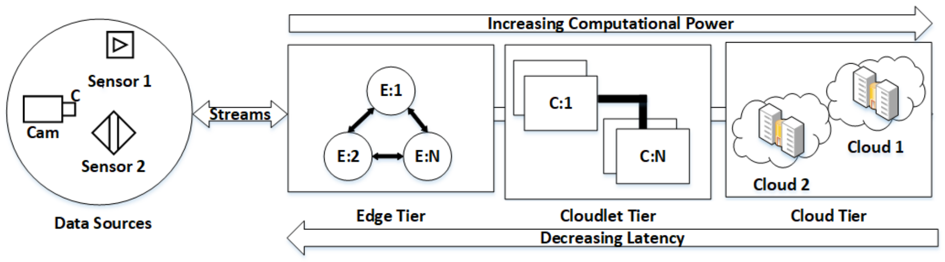


Fig. 3. Edge Enhanced Architecture for Stream Processing

4 Horizontal-Vertical (HV) Stream Analytics Architecture

We propose an architecture consisting of data sources, in-transit edge and fog nodes and clouds in three tiers namely edge, cloudlet and cloud as shown in Fig. 3. The first tier consists of edge resources and the second tier contains cloudlets, which are in-transit fog resources. The resources labelled as E:1, E:2 represent edge nodes 1 and 2 respectively and E:N represents the n^{th} edge node. Similarly, resources in the cloudlet tier are labelled as C:1, C:2, C:N which represent the corresponding cloudlet nodes 1-2 and the n^{th} cloudlet node respectively. The third tier consists of a federation of multiple clouds. Generally, resources at the same tier are assumed to reside at the same physical site. However, different tiers may reside at geographically different physical locations, and therefore may have varying interaction latencies.

Video data moves from the data source passing through the in-transit nodes to reach the destination cloud platform. Data stream elements are generated at regular intervals, with these elements being initially processed by the edge tier followed by the cloudlet and cloud tiers.

As we move from the edge to the cloud tier, the computational power of tiers increases from left to right. The edge tier has less computational power than the cloudlet or cloud tiers as edge resources have constrained storage and computational capacities compared to resources in the cloudlet or cloud tiers. Similarly, cloud resources have more storage and compute capacity than cloudlet or edge resources. Latency, which is the time taken by a frame to reach the computational resource, decreases from cloud to edge tier. Fig. 3 shows that there is a compromise between latency and computational capacity of a resource in the proposed three-tier architecture.

The proposed architecture is scalable to a large number of streams while still satisfying user QoS requirements. This is achieved by using multiple resources in each tier. The architecture can scale in the horizontal and vertical direction. For example in an edge tier, we can have one or more edge nodes to process the video streams in parallel which leads to vertical scalability. Similarly, one or more nodes can be connected in sequence to process the dependent stages leading to horizontal scalability.

4.1 Stages to Resource Mapping

The mapping of stages to resources is an important criterion which can affect the performance of the overall system. This mapping can be based on several criteria such as minimizing the power and latency [28]. It can also be based on reducing response time [29] or the total system energy cost [30] while satisfying user-defined QoS requirements. In contrast to these approaches, our resource allocation is based on a combination of resource latency and deadline time to maximize the analytics performance while using the minimum number of resources.

Algorithm 1 Stages to Resources Mapping

```
Input: Stages, resources, fps and JobdeadlineTime.
Output: Stages assigned to resources
    {Sort the resources using their latency}
 1: resources.sort(compare_latency)
 2: for all $stage$ in $stages$ do
 3:   stageAssigned=False
 4:   for all $res$ in $resources$ do
 5:     if (res.isReusable() or res.type="Cloud") then
 6:       parallelResCount=res.getParallelResources().size()
 7:       if (parallelResCount>0) then
 8:         stagesDivision=FPS/parallelResourcesCount
 9:         FPS=stagesDivision
10:       end if
11:       timeReq=res.getExecutionTime(stage,FPS)
12:       totalTimeReq=res.getPreviousTime()+timeReq
13:       if (totalTimeReq< jobDeadlineTime) then
14:         res.assignStage(stage)
15:         res.setPreviousTime(totalTimeReq)
16:         res.setReusable(True)
17:         stageAssigned=True
18:       else if (totalTimeRequired=jobDeadlineTime) then
19:         res.assignStage(stage)
20:         stageAssigned=True
21:         res.setReusable(False)
22:       else if (totalTimeReq>jobDeadlineTime) then
23:         res.setReusable(False)
24:       end if
25:       if (stageAssigned=False and res.type= "$Cloud$")
          then
26:         res.assignStage(stage)
27:       end if
28:     end if
29:   end for
30: end for
```

We propose an algorithm 1 to automatically assign stages to the number of participating resources in the proposed three tier architecture. The algorithms prefers low latency resources such as edge and in-transit resources and uses the cloud resources as a last resort to reduce the bandwidth and cost associated with the cloud platform. The algorithm is based on the two criteria.

- Latency: Resources which are near to the data source have less latency than resources which are far from

TABLE 2
Frame Record header

Size(Bytes)	Stage bits	Frame Data
N	N*8	(Variable)

it. Generally, the latency of edge resources is less than in-transit resources which have less latency than the cloud resources. Latency order is given by $E < I < C$ where E, I and C stands for the edge, in-transit and cloud resources. The intuition behind latency is that the closer the resource, the less time it takes for the data to reach the resource, resulting in early processing in contrast to propagating the data to (geographically) far resources such as a cloud platform.

- Deadline Time: Each job has a deadline by which it must be processed. For streaming applications, the deadline for real-time processing is often < 1s. If a stage takes more time to process than the deadline time on a resource, then it should not be allocated to that resource. The intuition behind deadline time is that streaming jobs should be processed before the next streaming job arrives at the network resource. Otherwise, more jobs will start accumulating at the resource queue buffer.

The algorithm 1 intelligently maps the stages to available resources. The algorithm has two parts. In the first part (step 1), we find all the available resources and sort them based on their latency. In the second part (steps 2-30), for each pipeline stage we find the most suitable resource by iterating over sorted resources from start to finish. The steps 5, determines if the resource is usable and steps 6-9 finds parallel resource(s) and divides the frame rate amongst them. Steps 13-24 have an IF-ELSE block which decides the assignment of stage to resources using three cases. For the first case (steps 13-17), if the total time required (TTR) is less than the job deadline time (JDT), then a stage is assigned and the resource is set reusable. For the second case (steps 18-21), if the TTR is equal to the JDT, then the stage is still assigned to the resource but the resource reusable flag is set to false. In the third case (steps 22-24), if the TTR is greater than the JDT, then the resource reusable flag is set to false, so it cannot be used for any more assignment of stages. The steps 25-27, checks if the stage has not been assigned to any edge or in-transit resource and assigns the stage to the cloud as a last resort. By default a resource is reusable. The algorithm can reuse a resource for multiple stages if the time required to process them does not exceed the deadline time. In cases where the frame rate (FPS) of a stage processing exceeds the deadline, it can make use of the vertical scalability for parallel processing as given in section 4.

4.2 Stage Tagging and Serialization

The video analytics problem is decomposed into stages which must be completed in a serial order to maintain their conceptual integrity and completeness. The dependent stages should be executed serially while independent stages can be executed in any order. To complete frame processing, each resource executes Algorithm 2 for each stage assigned

Algorithm 2 Frame Stages Synchronization

```
Input: A frame, stage number N
Output: Frame stage completed
    Initialization :
    Get the frame stage to execute
 1: stagePending=stage
    get a list of stage dependencies
 2: frameDependentStages=getStageDep(stagePending)
    Check for any unfinished stage
 3: time=0;
 4: for each stage in frameDependentStages do
 5:   if (frame.isStageCompleted(stage)=False) then
 6:     time=time+executeStage(stage)
 7:     if time>=jobDeadlineTime then
 8:       forwardFrameToNextNode(frame)
 9:       return
10:     end if
11:   end if
12: end for
    All dependent stages completed upto this point
13: status=executeStage(stagePending)
    if status indicates success, mark the frame stage as completed
14: if (status=True) then
15:   frame.setStageCompleted(stagePending)
16: end if
    Forward the frame to the next resource
17: forwardFrameToNextNode(frame)
```

to it. This algorithm takes a frame and a stage number N which has to be executed on the frame. In steps 1-2, it finds all dependent stages which must be completed to execute the stage N. In steps 4-12, it attempts to execute all dependent stages while continuously checking that the execution time does not exceed the deadline. If the execution time exceeds the deadline, it saves the current state of the frame and forwards the frame to the next node. In step 13, after execution of the dependent stages, it executes stage N. In steps 14-16 it checks for the execution status, if the execution is successful it marks the stage as completed and saves the frame state. In step 17, it forwards the frame to the next node for further processing (if necessary).

To satisfy the frame completion properties given in section 3, each frame job is tagged with metadata. Every frame is prefixed with an octet string, and the size of this string depends on the number of stages used. The minimum length of the tag is a single byte (8 bits) which can accommodate a max of 8 stages; there is no limit to the max number of stages. As the addition of a metadata tag to an input frame is an overhead, we designed the tag to occupy a minimum size. The tag size can be estimated as the number of total stages divided by eight. For example, overhead for 5 stages will be represented by an 8-bit string initialized to zero. As the frame moves across nodes, each resource node will activate the bit corresponding to its stage to indicate completeness of the stage. Both independent and dependent stage completeness is represented by a single bit. Each resource has access to a stage pipeline graph which the resource uses to turn on the bits in the tag. For instance, if a node is performing stage 3 then after processing has completed, and the frame modified, it will add string 00100000 to its header assuming other stages have not yet been processed. Each frame is prefixed with a frame record header as shown in Table 2.

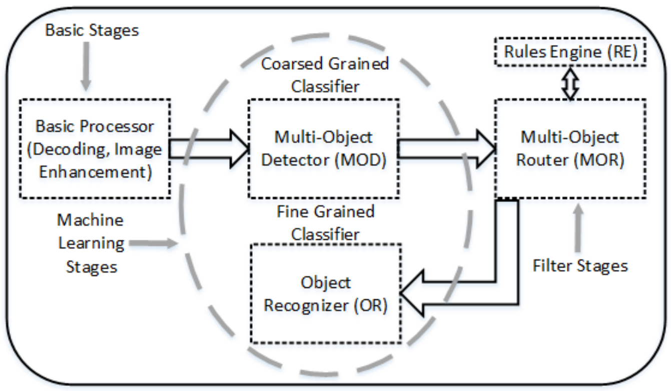

Basic Stages
Coarsed Grained Classifier
Rules Engine (RE)
Basic Processor (Decoding, Image Enhancement)
Multi-Object Detector (MOD)
Multi-Object Router (MOR)
Machine Learning Stages
Fine Grained Classifier
Filter Stages
Object Recognizer (OR)

Fig. 4. RealEdgeStream Components Dataflow with stage types

4.3 RealEdgeStream Components

In Fig.4, we show the main components of the system along with their dataflow and stage types associated with the components. These components may reside on one or more resource depending on the specific analytics problem. The basic processor is the first component which receives the input and performs the basic stages. It then sends the data to the Multi Object Detector (MOD) which performs object detection on the received data. The MOD is a coarse grained classifier to quickly detect one or more objects in the input frame using a machine learning model. The MOD takes video frames as input and detects the type of object using a You only look once (YOLO) detector [31] – a high performance classifier which can output an object into 80 classes. We have used YOLO V3 for detection of two classes namely person and vehicle.

The MOD component then extracts the bounding boxes of the detected objects and forwards them to the Multi Object Router (MOR) which decides which objects to send to the Object Recognizer (OR) based on rules. The purpose of the MOR is to perform filtration, optimized scheduling and routing of the detected object to the cloud. It loads a text file which specifies the type of white-list objects. The MOR only allows routing of white-listed objects, all other object type jobs are filtered and deleted at this node. The white-list can be changed by the administrator at run time to modify the behaviour of the network for different applications. The MOR uses a Rules Engine (RE) component to forward or reject frames in the sink. The MOR component performs the filter stages using configurables rules – identifying which types of objects should be forwarded to the Object Recognizer (OR). All other object types are rejected by this component. The MOR then forwards the data to the Object Recognizer (OR) component which has a fine grained classifier to perform further analysis on the data (using a deep learning model to recognize the object).

5 Video Stream Analytics Use Case

An object recognition scenario is used to demonstrate the proposed architecture from Section 4, as illustrated in Fig. 5. As video frames move from source to the cloud platform, they are processed by the edge and in-transit resources. The edge node executes the basic and filter stages on the video frame. Basic stages include frame loading and decoding, frame resizing, motion detection, and image enhancement. It also resizes the frame to a 224x224 resolution – the input resolution required for the deep learning object recognition model (for person and vehicle). An edge node filtration stage performs motion detection between frames.

At the cloudlet node, a machine learning stage performs object detection on the incoming frames and the filtration stage determines which frames to forward to the cloud platform. The cloudlet is equipped with a set of rules which define the behaviour of the filtration stage. If no object is detected or if the detected object is not in the user white-list rules, it is forwarded to a sink where it is rejected. For this case, we have programmed the cloudlet to only forward frames which contain a detected person or vehicle object. This can considerably save cloud bandwidth and processing time on the cloud. Finally, the recognition task is performed either by a cloud or a cloudlet node depending on the distribution of the stages by a placement algorithm as given in section 4.1.

5.1 Object Recognition Training and Inference Dataset

We trained two deep learning models for object recognition to support vehicle and person recognition. For vehicle recognition, we used the Stanford vehicle dataset [32]. The dataset consists of 8144 images of vehicles with manufacturer make, model and year and ≈4GB in size. We used the state of the art MobileNet [33] model to train on the dataset. MobileNet is an open source convolutional neural network model developed by Google. The MobileNet model is designed to maximize recognition performance while keeping a high accuracy. This is achieved by a depthwise separable convolution technique. It has demonstrated an accuracy of 70% on the ImageNet dataset. For our datasets, MobileNet achieved an accuracy of about ≈96% and ≈88% for vehicle and person recognition respectively. The model was fully trained after 2 hours and 10 mins on a Xeon E5 system with 8 cores and Nvidia GTX 970 GPU. The MobileNet model was chosen for its small size and low latency.

For person recognition, we used the modified CASIA-Webface dataset [34]. CASIA-Webface is the second largest public dataset available for facial recognition. Due to its large size, we trained the model on a subset of this data. Our dataset consists of 100 individual faces with an average of 100 training faces for each individual. MobileNet was used to train this model with parameters shown in Table 4.

For both models, the resolution of the images in the training data was fixed to 224x224 – the minimum resolution required by the two models. The fixed resolution for both models was necessary to reduce the frame size and to accurately classify incoming frame by the deep learning models. The image scaling is performed by earlier stages in the video analysis pipeline.

For inferencing we used the Tiny ImageNet dataset which consists of 10K images arranged into 200 classes. For each class, it provides 50 test images. We passed a complete set of this dataset for generating the experimental results in section 7. Before feeding the dataset, we replaced 1000 images by the Stanford *cars* images and 1000 images by the CASIA-Webface images. We did this to have equal probability of detection of each category – person and vehicles.

TABLE 3
Experimental Configuration

Configuration	Symbol	Placement Stages
Cloud-Only	C	Cloud=1,2,3,4
Edge-Cloud	EC	Edge=1,2 Cloud=3,4
Edge-Cloudlet-Cloud	ECC	Edge=1,2 Cloudlet=3, Cloud=4
Edge-Cloudlet-MultiResource	ECM	Edge=1,2 Cloudlet=3,4

TABLE 4
Training Parameters for the MobileNet Model

Parameter	**Vehicle Recognition Model**	**Person Recognition Model**
Dataset	Stanford Cars	CASIA-Webface
Dataset Size (Uncompressed)	3.7 GB	8.8 GB
classes	196	100(Manually Reduced)
Trained Model Size	26.3MB	25MB
Training Samples (Images)	8144	14816
Image Resolution	224x224	224x224
Training Accuracy	0.9648	0.8841
Loss	0.1148	0.3900
Epochs	50	10
Batch Size	30	30
Time to Train (GPU)	2hrs&10mins	1 hour

Table 4 shows the detailed parameters used for the training of vehicle and person recognition using the MobileNet model.

6 Experimental Setup and Configuration

The experimental setup consists of a network simulator using Google Remote Procedure Call (gRPC) – a framework more efficient than using the corresponding REST API [35]. The gRPC is used to connect the the edge/cloud implementation with the OMNet discrete event network simulator. In this way network resources in the simulator communicate with an actual deployed system to simulate a virtual network implementation.

A frame rate of 25-30 frames per second is required for capturing smooth motion from a camera. The video frame rate is very important for surveillance. If the frame rate is very high, it will need more storage and processing time but will not miss any event or action in the video. If the frame rate is low, processing and storage requirement will be less but frames may not be able to capture all the events or actions such as a person running or a drone tracking an object.

TABLE 5
FPS Modes

Time(s)	Mode	FPS
0-50	offpeak	15
50-100	peak	30
After 100	superpeak	100

TABLE 6
Computational Power and Resource Queue Buffer Size

Name	Computational Speed	Buffer Size(Number of Jobs)
Edge	X	5K
Cloudlet	13X	100K
Cloud	150x	Unbounded

The experimental setup as shown in Fig. 6 consists of two camera nodes as data sources, in-transit nodes and the destination cloud node. The network nodes are connected with each other using connections which have a delay and bandwidth as given in Table 7. The network nodes also have limited storage and computational capacity as given in Table 6. These parameters are based on realistic values and were chosen to simulate different experimental conditions and results.

EdgeSwitch receives data from the two cameras and forwards them to the local edge node. CloudletSwitch connects the edge node switch with the Cloudlet. The Cloudlet is an intermediate processing device which sits in-between the cloud and edge nodes. We performed experiments on the four configurations as given in Table 3. For each experiment, the stages were distributed based on placement algorithm 1. The experiments were conducted with varying input load – namely offpeak (15fps), peak (30 fps) and superpeak (100 fps) as given in Table 5.

In a cloud only configuration, all the stages are executed on a central cloud. In this configuration, there is no intermediate processing on edge or fog nodes and represents the traditional centralized analysis scenario. The cloud-only configuration acts as a benchmark for comparing the performance and bandwidth with the Edge-Cloud, Edge-Cloudlet-Cloud and Edge-Cloudlet-MultiResource configurations. As outlined in [18], we show that Edge-Cloudlet-Cloud with filtration is the most efficient configuration as compared to Cloud-only, Edge-Cloudlet, or Edge-Cloudlet-Cloud configurations. It has the advantage of both concurrent processing and low-value data filtration. This is achieved by utilizing in-transit resources to accelerate video processing. In Edge-Cloudlet-MultiResource, we filter and forward the incoming video streams to multiple resources based on the detected objects to satisfy real-time processing requirements. In the Edge-Cloudlet-Multi-Resource configuration, basic processing stages were executed on the edge node which was followed by a motion detection filtration stage. Similarly, the machine learning stages for object detection were executed on two cloudlets followed by an object filtration stage. Finally, object recognition is performed on the cloud platform for the filtered frames. This configuration uses all the in-transit nodes from the data source to the destination and routes the data to the cloud platform.

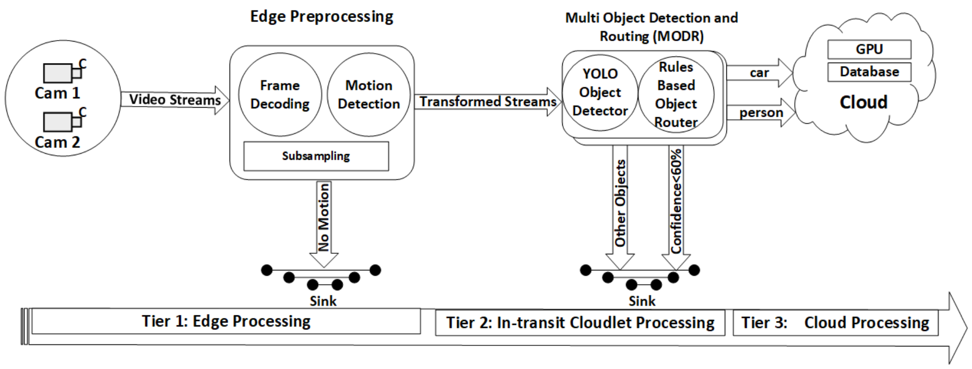


Fig. 5. Edge Enhanced Video Streams Analytics Architecture for Multi-Object Recognition

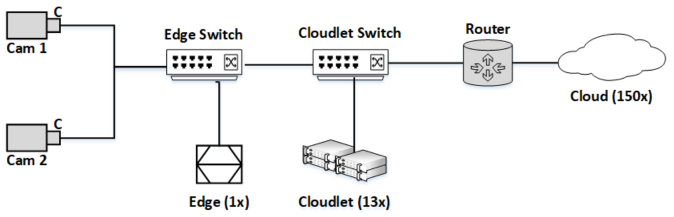


Fig. 6. Experimental setup for Multi-object recognition

7 Results and Discussion

7.1 Experiment 1: Time taken by individual stages

In this experiment, we measure the time taken by individual pipeline stages to complete frame processing (a job). The time taken by individual stages is important to efficiently distribute the stages on the available resources. It can be used to aggregate stages together or to decompose a stage into multiple stages. The time taken by stages 1-4 is shown in Fig. 7 for the incoming streams. The time taken by each individual stage to complete was plotted against the experiment elapsed time. The data points illustrated in the graph are average values for a number of executed jobs.

The time taken by each stage of a job is represented on the y-axis. The x-axis denotes the time elapsed during the experiment. Stages 4V and 4P indicate the time taken by the deep learning models to recognize the vehicle and person respectively. From the graph, jobs in stages 1 and 2 are completed in about 1ms, job execution in stages 3 and 4V varies from 200ms to 270ms. Jobs in Stage 4P take around 150ms on average.

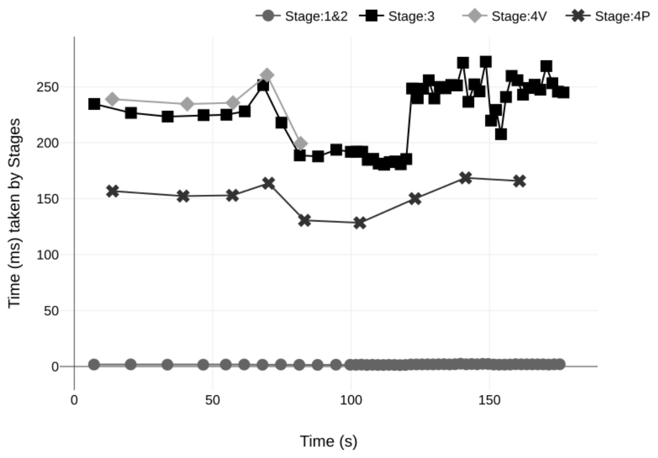


Fig. 7. Time taken by stages 1-4

7.2 Experiment 2: Number of Ready-Queue Jobs

Incoming streams are first buffered in a ready-queue at each resource. In this experiment, we counted the number of jobs in a ready-queue which are ready to be executed but are waiting for the resource to be allocated to them. This experiment shows the load on a particular resource and can be used to identify the bottleneck resource to satisfy QoS requirements of a user. A resource with very high ready-queue jobs means it is not able to process all of the jobs that were produced per unit time e.g. within 1 second by the data source. On the other hand, resources with a low number of ready-queue jobs indicate a resource which is able to process all of the incoming jobs in its ready-queue.

Fig. 8 shows the number of ready-queue jobs at the edge, cloudlet and cloud resources for the configurations given in Table 3. In the figure, a resource is prefixed with the configuration symbol, to differentiate between the same resource queues. For the cloudlet resource in the Edge-Cloudlet-Cloud configuration, we see a constant horizontal line for 15fps and 30fps indicating that the resource is able to process all of the jobs produced at that frame rate. After t=100 seconds when the frame rate equals 100fps, the cloudlet resource queue rises sharply and continues to show a steep upwards trend indicating the resource is unable to process the frame rate in real-time. In this case, the ready-queue size may exceed the cloudlet buffer limit given in Table 6 at some point. After this limit has been exceeded, the default behaviour is to remove the job which has been in the queue the longest and forward it to the next node without

TABLE 7
Delay and Bandwidth

Name	Average Delay(ms)	Max Uplink Bandwidth
Cam-Edge	1	1 Gbps
Edge-Cloudlet	2	1 Gbps
Cloudlet-Cloud	6	40 Mbps

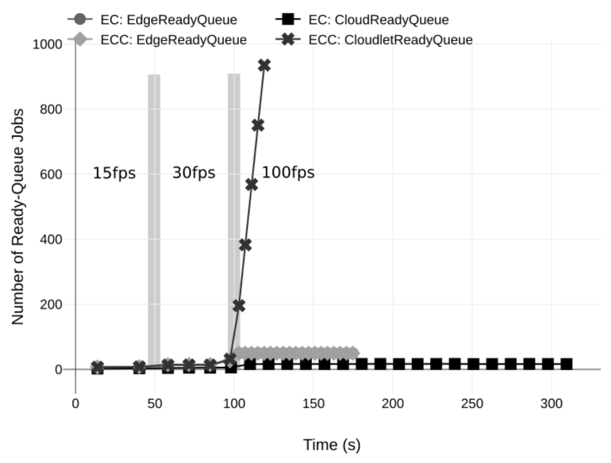


Fig. 8. Number of Ready-Queue Jobs

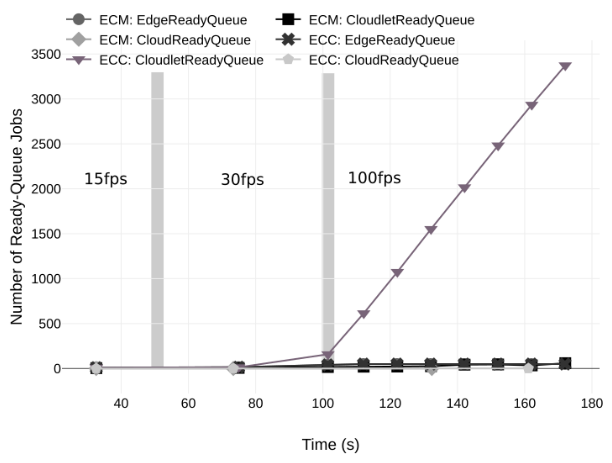


Fig. 9. Number of Ready-Queue Jobs with ECM

processing, which makes additional room for an incoming job. This process is repeated for any new jobs where a buffer is not able to hold more jobs.

It can be seen for other resource queues that the number of ready-queue jobs is very low, indicating that these resources are able to process all the incoming jobs as soon as they arrive. From this experiment, we conclude that none of the configuration namely Cloud, Edge-Cloud, Edge-Cloudlet-Cloud is able to process the jobs in real time after t=100 seconds. These configurations are only able to satisfy QoS constraints when the frame-rate is less than or equal to 30fps.

As proposed in Section 4, the horizontal and vertical scalability allows us to meet the QoS demand by employing multiple edge, cloudlet and cloud resources. The complexity of adding a new resource increases from edge to cloud tier. Edge and cloudlet being the local resources are easier to add and deploy than cloud resources. In this case, the slow performance is a direct result of the cloudlet node being unable to process all the frames when the frame rate approaches 100fps. We introduce a new configuration ECM, which is a variant of ECC and includes two cloudlet resources. It can be seen from Fig. 9 that after adding a second cloudlet resource, the cloudlet queue in ECM case goes down and becomes a constant – indicating the multi-resource approach is able to process all the streaming data produced at 100fps. However even if all of the resources are able to process the data in real-time, the job completion time is still limited by the uplink cloud bandwidth which in most cases is insufficient to process the data at 100fps. In case of ECM configuration with two cloudlet nodes, the placement algorithm discussed in section 4.1 is able to place both stages 3 and 4 on cloudlet resources. In this case, only the results from stage 4 were sent to the cloud for statistics and record keeping purpose.

7.3 Experiment 3: Job Execution Time

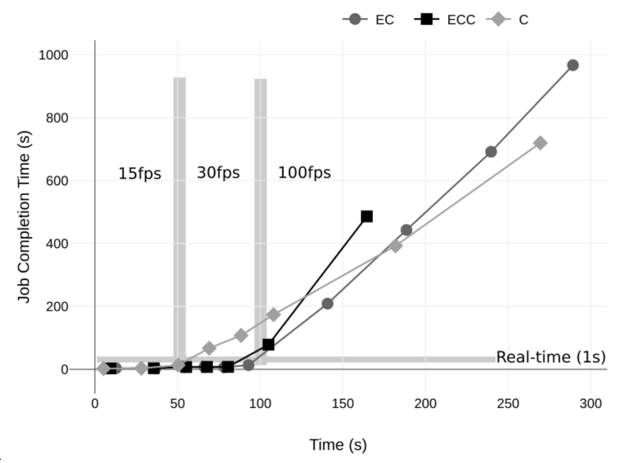
In this experiment we measure the time taken for a single job completion. This is the time difference between job creation and completion. The job execution time was noted for three configurations as given in Table 3 with the exclusion of ECM configuration. We note the difference between the job creation time and job completion time to plot the curve as shown in Fig. 10. Ideally, for a real-time requirement, all the streaming jobs produced per unit time e.g. one second, should be completed before additional streaming jobs arrive at the resource. Interestingly, this graph shows that the three configurations can support real-time completion of jobs for 15 frames per second. Real-time job completion is the time taken to complete a job in less than 1 second. However after t=50 seconds when the frame rate becomes 30fps, we can see a slight divergence from the real-time job completion line for the three configurations. The time to complete a single job after t=50 seconds increases rapidly which is shown by a steep upward curve for a cloud-only configuration.

It can be seen that after t=50 seconds when the frame rate equals 100fps, none of the three configurations are able to complete job processing in real time. By employing the ECM configuration with two cloudlet nodes as in experiment 2, it can be seen in Fig.11 that the ECM approach shows a linear curve which remains constant during the full experiment, indicating this configuration is able to process all the data in real time for 100fps.

7.4 Experiment 4: Jobs completed in a window time of 10, 60, 120 seconds

In this experiment, we noted the number of jobs completed in a window time of 10, 60 and 120 seconds on the configurations given in Table 3. We skipped the Edge-Cloud configuration as it yielded the same result as the central cloud. We vary the frame rate of cameras from 15-100 frames per second. A variable frame rate is important to compare the job completion per unit time for the three configurations. The results are shown in Fig. 12.

For the 10 seconds window, the Cloud-Only configuration is able to process more jobs than the other configuration. This is due to only 15 frames per second being produced in the first 50 seconds which does not exceed the uplink bandwidth of the cloud. After 50 seconds, the total frame rate from both cameras is increased to 30 fps and we can



.

Fig. 10. Single Job Completion Time

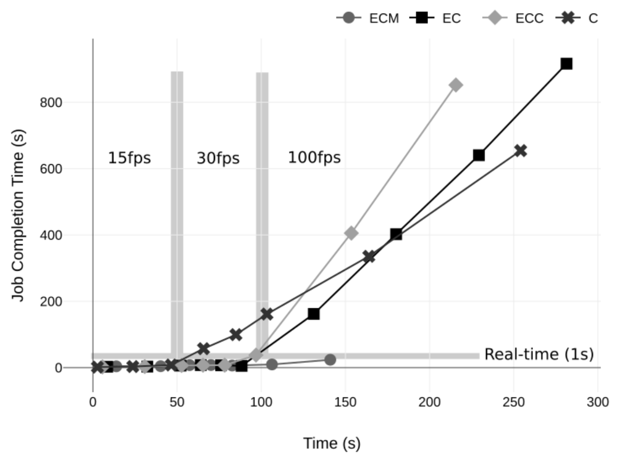


Fig. 11. Single Job Completion Time

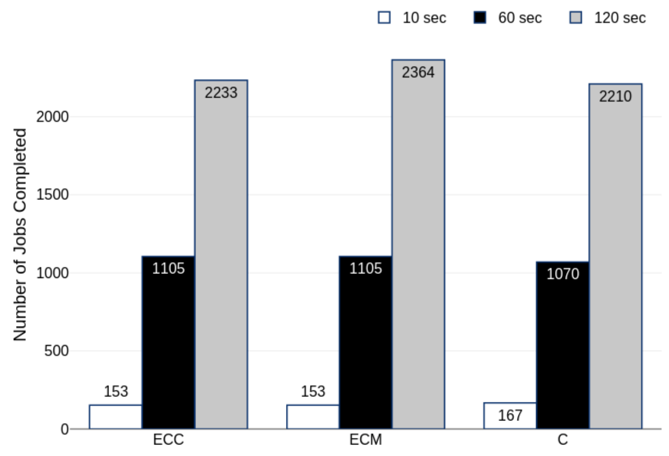


Fig. 12. Number of Jobs Processed in 10, 60 and 120 seconds

see that ECC and ECM configurations are able to process more jobs than the CO configuration due to the limited bandwidth of the cloud. For the 120 seconds window, when 100 frames are being produced in one second, all the three configurations are operating in the superpeak mode and we see a different number of jobs being completed for each of the configurations. Edge-Cloudlet-MultiResource has the highest number of jobs completed, whereas the second highest number of jobs completed is in the Edge-Cloudlet-Cloud configuration.

7.5 Experiment 5: Time taken to complete all jobs

In this experiment, the time taken to complete all jobs was noted for the three configurations given in Table 3 and a graph was plotted as shown in Fig. 13. It can be seen in the figure that for the offpeak mode which only produces 15 frames for the first 50 seconds, all the configurations complete the same number of jobs. After 50 seconds, the camera switches to peak mode and we start to see a divergence in the three curves of the graph. After 100 seconds in superpeak mode, the divergence becomes more prominent and steep which increases the frame rate to 100 per second. The real-time finish line indicates the real-time requirement for the experiment.

At t=177 seconds, both the cameras have finished transmitting their data. For real-time, all the resources must complete the data processing in 177 seconds. However, there is an overhead of 173 seconds for the cloud-only case and around 58 seconds for the Edge-Cloudlet-Cloud case. Only Edge-Cloudlet-MultiResource configuration is able to complete all the jobs within 177 seconds. This means all jobs were processed by this configuration as soon as they were produced.

It can be seen that the time taken to complete all jobs is highest for the cloud only configuration as all the streaming data produced per unit time is transferred to the cloud first and then analytics take place. The Edge-Cloudlet-MultiResource is the most efficient configuration which is able to complete the processing of all the jobs in real time. Using percentage gain from equation 8, this ECM configuration is 49% more efficient in terms of time taken than the cloud-only configuration.

7.6 Experiment 6: Bandwidth Consumed

In this experiment, we measured the bandwidth consumed from the data source to cloud for the configurations given in Table. 3. The bandwidth consumed varies over time due to a change in frame rate from 15 to 100 frames per second. The graph in Fig. 14 shows the available bandwidth and consumed bandwidth for all of the configurations. It can be seen that for the first 50 seconds, the bandwidth consumed by all configurations is below 20Mbps, however after 50 seconds when the camera is operating in peak mode, the required bandwidth rises sharply and approaches the available bandwidth for Cloud-only and Edge-Cloudlet case. After 100 seconds, when the camera switches to superpeak mode, the bandwidth required by the Cloud-Only and Edge-Cloud configuration exceeds the available bandwidth of 40Mbps.

It can also be seen that the bandwidth required by the ECC and ECM configurations is significantly less than the

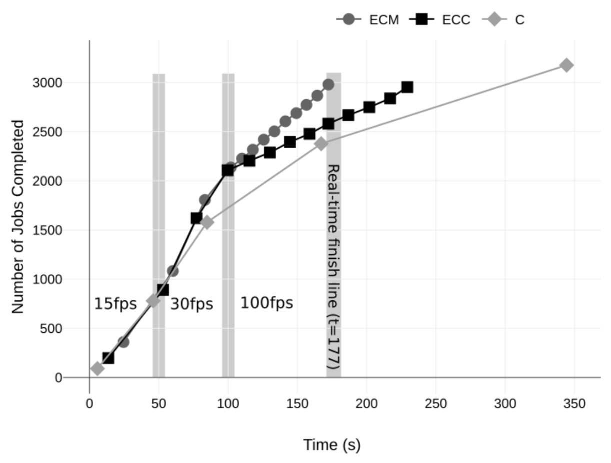


Fig. 13. Total jobs completed in time

other configurations. This is due to both ECC and ECM configurations performing filtering of the low-value data and only sending the bounding boxes of the detected objects which reduces the bandwidth. The ECM configuration further optimizes this by only sending the results of stage 4 to the cloud. In ECM case, both stages 3 and 4 are executed on the cloudlet.

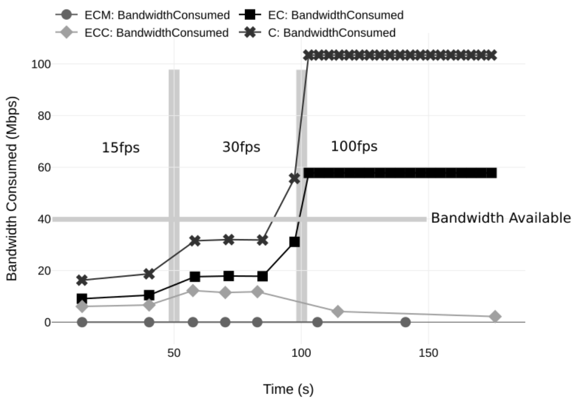


Fig. 14. Bandwidth consumed over time

Overall, for our dataset, the ECM configuration saves 99% bandwidth than the cloud-only case.

8 Conclusions

We investigate an architectural approach for supporting scalable real-time video stream processing using edge and in-transit computing. Current approaches to stream analytics are based on the use of centralized cloud platforms. With the increase in data volumes and velocity, a centralised analysis approach of this kind becomes infeasible for high-performance applications due to limited uplink bandwidth, variable latency and congestion between the data sources and the cloud platform.

We propose RealEdgeStream (RES), an edge enhanced stream analysis system which complements a cloud-based platform to enable real-time processing of video streams. Our approach to video analysis consists of a filtration phase followed by an identification phase. The filtration phase allows objects of low-value to be filtered (discarded) by the edge and in-transit nodes using configurable rules from a user. The identification phase performs deep learning inference on the objects of interest. It can be used to perform further complex analytics such as pattern, object and activity recognition. The phases consist of three types of stages namely basic, filter and machine learning to logically analyze and partition the video analysis pipeline. The system intelligently distributes processing stages on the available resources using an algorithm to satisfy user Quality of Service requirements.

Our experimental results show that scalable real-time performance can be achieved by a fair use of multiple resources in the proposed three-tier architecture. The results show that the Edge-Cloudlet-MultiResource (ECM) configuration is able to achieve real-time performance for 100 frames per second when all the stages are placed on the edge and in-transit resources as compared to only 15 frames per second using the cloud-only approach. For the experimental dataset consisting of 10K data streams, it takes 49% less time to complete and also saves 99% of the bandwidth consumed as compared to a centralized cloud based analytics approach.

Acknowledgments

This research is supported, in part, by the U.S. National Science Foundation (NSF) under grants numbers OAC 1640834, OAC 1835661, OAC 1835692, OCE 1745246. We also thank Danielle Turvill for her support in reviewing this paper.

References

[1] C. Shan, F. Porikli, T. Xiang, and S. Gong, *Video Analytics for Business Intelligence*. Springer, 2012.

[2] M. U. Yaseen, A. Anjum, O. Rana, and R. Hill, "Cloud-based scalable object detection and classification in video streams," *Future Generation Computer Systems*, vol. 80, pp. 286–298, 2018.

[3] T. Abdullah, A. Anjum, M. F. Tariq, Y. Baltaci, and N. Antonopoulos, "Traffic monitoring using video analytics in clouds," in *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE Computer Society, 2014, pp. 39–48.

[4] Y.-L. Chen, T.-S. Chen, T.-W. Huang, L.-C. Yin, S.-Y. Wang, and T.-c. Chiueh, "Intelligent urban video surveillance system for automatic vehicle detection and tracking in clouds," in *2013 IEEE 27th international conference on advanced information networking and applications (AINA)*. IEEE, 2013, pp. 814–821.

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

[6] W. Zhang, L. Xu, P. Duan, W. Gong, Q. Lu, and S. Yang, "A video cloud platform combing online and offline cloud computing technologies," *Personal and Ubiquitous Computing*, vol. 19, no. 7, pp. 1099–1110, 2015.

[7] R. Pereira, M. Azambuja, K. Breitman, and M. Endler, "An architecture for distributed high performance video processing in the cloud," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE, 2010, pp. 482–489.

[8] K. Shvachko, H. Kuang, S. Radia, R. Chansler *et al.*, "The hadoop distributed file system." in *MSST*, vol. 10, 2010, pp. 1–10.

[9] C. Ryu, D. Lee, M. Jang, C. Kim, and E. Seo, "Extensible video processing framework in apache hadoop," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 2. IEEE, 2013, pp. 305–310.

[10] A. Kafka, "A high-throughput, distributed messaging system," *URL: kafka. apache. org as of*, vol. 5, no. 1, 2014.

[11] A. Spark, "Apache spark: Lightning-fast cluster computing," *URL http://spark. apache. org*, 2016.

[12] M. H. Iqbal and T. R. Soomro, "Big data analysis: Apache storm perspective," *International journal of computer trends and technology*, vol. 19, no. 1, pp. 9–14, 2015.

[13] M. U. Yaseen, A. Anjum, M. Farid, and N. Antonopoulos, "Cloud-based video analytics using convolutional neural networks," *Software: Practice and Experience*, vol. 49, no. 4, pp. 565–583, 2019.

[14] A. Anjum, T. Abdullah, M. Tariq, Y. Baltaci, and N. Antonopoulos, "Video stream analysis in clouds: An object detection and classification framework for high performance video analytics," *IEEE Transactions on Cloud Computing*, 2016.

[15] M. U. Yaseen, A. Anjum, O. Rana, and N. Antonopoulos, "Deep learning hyper-parameter optimization for video analytics in clouds," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, no. 99, pp. 1–12, 2018.

[16] 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 *NSDI*, vol. 9, 2017, p. 1.

[17] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos, "Edge analytics in the internet of things," *IEEE Pervasive Computing*, vol. 14, no. 2, pp. 24–31, 2015.

[18] M. Ali, A. Anjum, M. U. Yaseen, A. R. Zamani, D. Balouek-Thomert, O. Rana, and M. Parashar, "Edge enhanced deep learning system for large-scale video stream analytics," in *Fog and Edge Computing (ICFEC), 2018 IEEE 2nd International Conference on*. IEEE, 2018, pp. 1–10.

[19] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *computer*, vol. 50, no. 10, pp. 58–67, 2017.

[20] A. R. Zamani, M. Zou, J. Diaz-Montes, I. Petri, O. Rana, A. Anjum, and M. Parashar, "Deadline constrained video analysis via in-transit computational environments," *IEEE Transactions on Services Computing*, 2017.

[21] T. Zhang, A. Chowdhery, P. V. Bahl, K. Jamieson, and S. Banerjee, "The design and implementation of a wireless video surveillance system," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM, 2015, pp. 426–438.

[22] C. Ballas, M. Marsden, D. Zhang, N. E. O'Connor, and S. Little, "Performance of video processing at the edge for crowd-monitoring applications," 2018.

[23] P. Liu, B. Qi, and S. Banerjee, "Edgeeye: An edge service framework for real-time intelligent video analytics," in *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking*. ACM, 2018, pp. 1–6.

[24] E. G. Renart, D. Balouek-Thomert, and M. Parashar, "An edge-based framework for enabling data-driven pipelines for iot systems," in *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2019, pp. 885–894.

[25] C.-C. Hung, G. Ananthanarayanan, L. Golubchik, M. Yu, and M. Zhang, "Wide-area analytics with multiple resources," in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, p. 12.

[26] Q. Zhang, H. Sun, X. Wu, and H. Zhong, "Edge video analytics for public safety: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1675–1696, 2019.

[27] M. Proctor, "Drools: a rule engine for complex event processing," in *Proceedings of the 4th international conference on Applications of Graph Transformations with Industrial Relevance*. Springer-Verlag, 2011, pp. 2–2.

[28] A. Mukherjee, D. De, and D. G. Roy, "A power and latency aware cloudlet selection strategy for multi-cloudlet environment," *IEEE Transactions on Cloud Computing*, vol. 7, no. 1, pp. 141–154, 2016.

[29] X. Sun and N. Ansari, "Latency aware workload offloading in the cloudlet network," *IEEE Communications Letters*, vol. 21, no. 7, pp. 1481–1484, 2017.

[30] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1171–1181, 2016.

[31] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[32] "Stanford cars dataset," 2020 (accessed May 6, 2020). [Online]. Available: https://ai.stanford.edu/~jkrause/cars/car_dataset.html

[33] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[34] D. Yi, Z. Lei, S. Liao, and S. Z. Li, "Learning face representation from scratch," *CoRR*, vol. abs/1411.7923, 2014. [Online]. Available: http://arxiv.org/abs/1411.7923

[35] S. G. Du, J. W. Lee, and K. Kim, "Proposal of grpc as a new northbound api for application layer communication efficiency in sdn," in *Proceedings of the 12th International Conference on Ubiquitous Information Management and Communication*. ACM, 2018, p. 68.

Muhammad Ali is currently working toward the PhD degree at the University of Derby, UK. He has experience of working in diverse tools, technologies and programming languages. He has also worked in the software industry for about 5 years. His research interests include edge and fog computing, big data analytics, deep learning, intelligent multi-agents, high-performance computing and video analytics.

Ashiq Anjum is a professor of distributed systems at the University of Leicester, UK. Previously, he was a professor of distributed systems and director of the Data Science Research Centre at the University of Derby, UK. His research interests are in data intensive distributed systems and high performance analytics platforms for continuous processing of streaming data. He has been consistently securing grants from different funding agencies for investigating high performance analytics systems for producing intelligence and evidence for engineering, aerospace, medical and security applications.

Omer Rana received the Ph.D. degree in neural computing and parallel architectures from Imperial College of Science, Technology and Medicine. He is a Professor of performance engineering at Cardiff University, U.K. His research interests include problem solving environments for computational science and commercial computing, data analysis and management for large-scale computing, and high performance distributed computing (including edge and cloud computing).

Ali Reza Zamani received the PhD degree in computer science from Rutgers University, New Brunswick, New Jersey. He is currently working as a Systems and Infrastructure Engineer at LinkedIn, California, USA. He completed his PhD in Computer Science from Rutgers University, US. His research interests are in the areas of Software Defined Networking(SDN), Network Functions Virtualization(NFV) and apply them in order to improve the performance of cloud federation systems

Daniel Balouek-Thomert received the PhD degree from the Ecole Normale Superieure de Lyon, France. He is currently is a Research Associate at the Rutgers Discovery Informatics Institute (RDI2) at Rutgers University in New Brunswick, New Jersey. His research interests lies in distributed computing, including models and middleware for Edge computing and stream processing applications. He received his PhD from the Ecole Normale Superieure de Lyon (France).

Manish Parashar is distinguished Professor of Computer Science at Rutgers University, New Brunswick, New Jersey. He is also the founding director of the Rutgers Discovery Informatics Institue (RDI2). His research interests are in the broad areas of parallel and distributed Computing and computational and data-Enabled science and sngineering. He is the founding chair of the IEEE Technical Consortium on High Performance Computing (TCHPC), editor-in-chief of the *IEEE Transactions on Parallel and Distributed Systems*. He is Fellow of AAAS, Fellow of IEEE/IEEE Computer Society and ACM Distinguished Scientist. For more information please visit http://parashar.rutgers.edu/