Video Stream Analysis in Clouds: An Object Detection and Classification Framework for High Performance Video Analytics

Ashiq Anjum, Tariq Abdullah, M. Fahim Tariq, Yusuf Baltaci, and Nick Antonopoulos

Abstract—Object detection and classification are the basic tasks in video analytics and become the starting point for other complex applications. Traditional video analytics approaches are manual and time consuming. These are subjective due to the very involvement of human factor. We present a cloud based video analytics framework for scalable and robust analysis of video streams. The framework empowers an operator by automating the object detection and classification process from recorded video streams. An operator only specifies an analysis criteria and duration of video streams to analyse. The streams are then fetched from a cloud storage, decoded and analysed on the cloud. The framework executes compute intensive parts of the analysis to GPU powered servers in the cloud. Vehicle and face detection are presented as two case studies for evaluating the framework, with one month of data and a 15 node cloud. The framework reliably performed object detection and classification on the data, comprising of 21,600 video streams and 175 GB in size, in 6.52 hours. The GPU enabled deployment of the framework took 3 hours to perform analysis on the same number of video streams, thus making it at least twice as fast than the cloud deployment without GPUs.

Index Terms—Cloud Computing, video stream analytics, object detection, object classification, high performance

1 Introduction

Recent past has observed a rapid increase in the availability of inexpensive video cameras producing good quality videos. This led to a widespread use of these video cameras for security and monitoring purposes. The video streams coming from these cameras need to be analysed for extracting useful information such as object detection and object classification. Object detection from these video streams is one of the important applications of video analysis and becomes a starting point for other complex video analytics applications. Video analysis is a resource intensive process and needs massive compute, network and data resources to deal with the computational, transmission and storage challenges of video streams coming from thousands of cameras deployed to protect utilities and assist law enforcement agencies.

There are approximately six million cameras in the UK alone [1]. Camera based traffic monitoring and enforcement of speed restrictions have increased from just over 300,000 in 1996 to over two million in 2004 [2]. In a traditional video analysis approach, a video stream coming from a monitoring camera is either viewed live or is recorded on a bank of DVRs or computer HDD for later processing. Depending upon the needs, the recorded video stream is retrospectively analyzed by the operators. Manual analysis of the recorded video streams is an expensive undertaking. It is not only time consuming, but also requires a large number of staff, office work place and resources. A human operator loses concentration from video monitors only after 20 minutes [3]; making it impractical to go through the recorded videos in a time constrained scenario. In real life, an operator may have to juggle between viewing live and recorded video contents while searching for an object of interest, making the situation a lot worse especially when resources are scarce and decisions need to be made relatively quicker.

Traditional video analysis approaches for object detection and classification such as color based [4], adaptive background [5], template matching [6] and Guassian [7] are subjective, inaccurate and at times may provide incomplete monitoring results. There is also a lack of object classification in these approaches [4], [7]. These approaches do not automatically produce colour, size and object type information [5]. Moreover, these approaches are costly and time consuming to such an extent that their usefulness is sometimes questionable [6].

To overcome these challenges, we present a cloud based video stream analysis framework for object detection and classification. The framework focuses on building a scalable and robust cloud computing platform for performing automated analysis of thousands of recorded video streams with high detection and classification accuracy.

An operator using this framework, will only specify the analysis criteria and the duration of video streams to analyse. The analysis criteria defines parameters for detecting objects of interests (face, car, van or truck) and size/colour

• *A. Anjum and N. Antonopoulos are with the College of Engineering and Technology, University of Derby, United Kingdom. E-mail: {a.anjum, n.antonopoulos}@derby.ac.uk.*
• *T. Abdullah is with the College of Engineering and Technology, University of Derby, United Kingdom and the XAD Communications, Bristol, United Kingdom. E-mail: t.abdullah@derby.ac.uk.*
• *M.F. Tariq and Y. Baltaci are with the XAD Communications, Bristol, United Kingdom. E-mail: {m.f.tariq, yusuf.baltaci}@xadco.com.*

Manuscript received 2 May 2015; revised 1 Oct. 2015; accepted 10 Dec. 2015. Date of publication 13 Jan. 2016; date of current version 29 Nov. 2019. Recommended for acceptance by P. Corcoran. Digital Object Identifier no. 10.1109/TCC.2016.2517653

based classification of the detected objects. The recorded video streams are then automatically fetched from the cloud storage, decoded and analysed on cloud resources. The operator is notified after completion of the video analysis and the analysis results can be accessed from the cloud storage.

The framework reduces latencies in the video analysis process by using GPUs mounted on compute servers in the cloud. This cloud based solution offers the capability to analyse video streams for on-demand and on-the-fly monitoring and analysis of the events. The framework is evaluated with two case studies. The first case study is for vehicle detection and classification from the recorded video streams and the second one is for face detection from the video streams. We have selected these case studies for their wide spread applicability in the video analysis domain.

The following are the main contributions of this paper: Firstly, to build a scalable and robust cloud solution that can perform quick analysis on thousands of stored/recorded video streams. Secondly, to automate the video analysis process so that no or minimal manual intervention is needed. Thirdly, to achieve high accuracy in object detection and classification during the video analysis process. This work is an extended version of our previous work [8].

The rest of the paper is organized as followed: The related work and state of the art are described in Section 2. Our proposed video analysis framework is explained in Section 3. This section also explains different components of our framework and their interaction with each other. Porting the framework to a public cloud is also discussed in this section. The video analysis approach used for detecting objects of interest from the recorded video streams is explained in Section 4. Section 5 explains the experimental setup and Section 6 describes the evaluation of the framework in great detail. The paper is concluded in Section 7 with some future research directions.

2 Related Work

Quite a large number of works have already been completed in this field. In this section, we will be discussing some of the recent studies defining the approaches for video analysis as well as available algorithms and tools for cloud based video analytics. An overview of the supported video recording formats is also provided in this section. We will conclude this section with salient features of the framework that are likely to bridge the gaps in existing research.

2.1 Object Detection Approaches

Automatic detection of objects in images/video streams has been performed in many different ways. Most commonly used algorithms include template matching [6], background separation using Gaussian Mixture Models (GMM) [9] and cascade classifiers [10]. Template matching techniques find a small part of an image that matches with a template image. A template image is a small image that may match to a part of a large image by correlating it to the large image. Template matching is not suitable in our case as object detection is done only for pre-defined object features or templates.

Background separation approaches separate foreground and background pixels in a video stream by using GMM [9]. A real time approximation method that slowly adapts to the values from the Gaussians and also deals with the multi-model distributions caused by several issues during an analysis has been proposed in [9]. Background separation methods are not suitable in our case as these are computationally expensive. Feature extraction using support vector machines [11] provided good performance on large scale image datasets with increasing number of nodes.

A cascade of classifiers (termed as HaarCascade Classifier) [10] is an object detection approach and uses real AdaBoost [12] algorithm to create a strong classifier from a collection of weak classifiers. Building a cascade of classifiers is a time and resource consuming process. However, it increases detection performance and reduces the computation power needed during the object detection process. We used a cascade of classifiers for detecting faces/vehicles in video streams for the results reported in this paper.

2.2 Video Analytics in the Clouds

Large systems usually consist of hundreds or even thousands number of cameras covering over wide areas. Video streams are captured and processed at the local processing server and are later transferred to a cloud based storage infrastructure for a wide scale analysis. Since, enormous amount of computation is required to process and analyze the video streams, high performance and scalable computational approaches can be a good choice for obtaining high throughputs in a short span of time. Hence, video stream processing in the clouds is likely to become an active area of research to provide high speed computation at scale, precision and efficiency for real world implementation of video analysis systems. However, so far major research focus has been on efficient video content retrieval using Hadoop [13], encoding/decoding [14], distribution of video streams [15] and on load balancing of computing resources for on-demand video streaming systems using cloud computing platforms [15].

Video analytics have also been the focus of commercial vendors. Vi-System [16] offers an intelligent surveillance system with real time monitoring, tracking of an object within a crowd using analytical rules. Vi-System does not work for recorded videos, analytics rules are limited and need to be defined in advance. Project BESAFE [17] aimed for automatic surveillance of people and tracking their abnormal behaviour using trajectories approach. It lacks scalability to a large number of streams and requires high bandwidth for video stream transmission.

IVA 5.60 [18] is an embedded video analysis system and is capable of detecting, tracking and analyzing moving objects in a video stream. EptaCloud [19] extends the functionality provided by IVA 5.60 in a scalable environment. The video analysis system is built into the cameras of IVA 5.60 that increases its installation cost. Intelligent Vision is not scalable and does not serve our requirements.

Because of abundant computational power and extensive support on multi-threading, GPUs have become an active research area to improve performance of video processing algorithms. For example, Lui et al. [20] proposed a hybrid parallel computing framework based on the MapReduce [21]. The results suggest that such a model will be hugely beneficial for video processing and real time video analytics systems. We aim to use a similar approach in this research.

TABLE 1
Supported Video Recording Formats

Video Format	Frame Rate	Pixels per Frame	Video Resolution	Video Size
CIF	25	99.0 K	352 × 288	7.50 MB
QCIF	25	24.8 K	176 × 144	2.95 MB
4CIF	25	396 K	704 × 576	8.30 MB
Full HD	25	1.98 M	1,920 × 1,080	9.35 MB

Existing cloud based video analytics approaches do not support recorded video streams [16] and lack scalability [17]. GPU based approaches are still experimental [20]. IVA 5.60 [18] supports only embedded video analytics, otherwise their approaches are close to the approach presented in this research.

The framework being reported in this paper uses GPU mounted servers in the cloud to capture and record video streams and to analyse the recorded video streams using a cascade of classifiers for object detection.

2.3 Supported Video Formats

CIF, QCIF, 4CIF and Full HD video formats are supported for video stream recording in the presented framework. The resolution of a video stream in CIF format is 352 × 288 and each video frame has 99 k pixels. Quarter CIF (QCIF) is a low resolution video format and is used in setups with limited network bandwidth. Video stream resolution in QCIF format is 176 × 144 and each video frame has 24.8 k pixels. The resolution of a video stream in 4CIF format is 704 × 576 and each frame has 396 k pixels. CIF and 4CIF formats have been used for acquiring video streams from the camera sources for traffic/object monitoring in our framework. Full HD (Full High Definition) video format captures video streams with 1,920 × 1,080 resolution. The video streams are captured at a constant bitrate of 200 kbps and at 25 fps in the results reported in this paper. Table 1 summarizes the supported video formats and their parameters.

3 Video Analysis Framework

This section outlines the proposed framework, its different components and the interaction between them (Fig. 1). The proposed framework provides a scalable and automated solution for video stream analysis with minimum latencies and user intervention. It also provides capability for video stream capture, storage and retrieval. This framework makes the video stream analysis process efficient and reduces the processing latencies by using GPU mounted servers in the cloud. It empowers a user by automating the process of identifying and finding objects and events of interest. Video streams are captured and stored in a local storage from a cluster of cameras that have been installed on roads/buildings for the experiments being reported in this paper. The video streams are then transferred to a cloud storage for further analysis and processing. The system architecture of the video analysis framework is depicted in Fig. 1 and the video streams analysis process on an individual compute node is depicted in Fig. 2a. We explain the framework components and the video stream analysis process in the remainder of this section.

3.1 Automated Video Analysis

The framework automates the video stream analysis by reducing the user interaction during this process. An operator/user initiates the video stream analysis by defining an "Analysis Request" from the APS Client component (Fig. 1) of the framework. The analysis request is sent to the cloud data center for analysis and no more operator interaction is

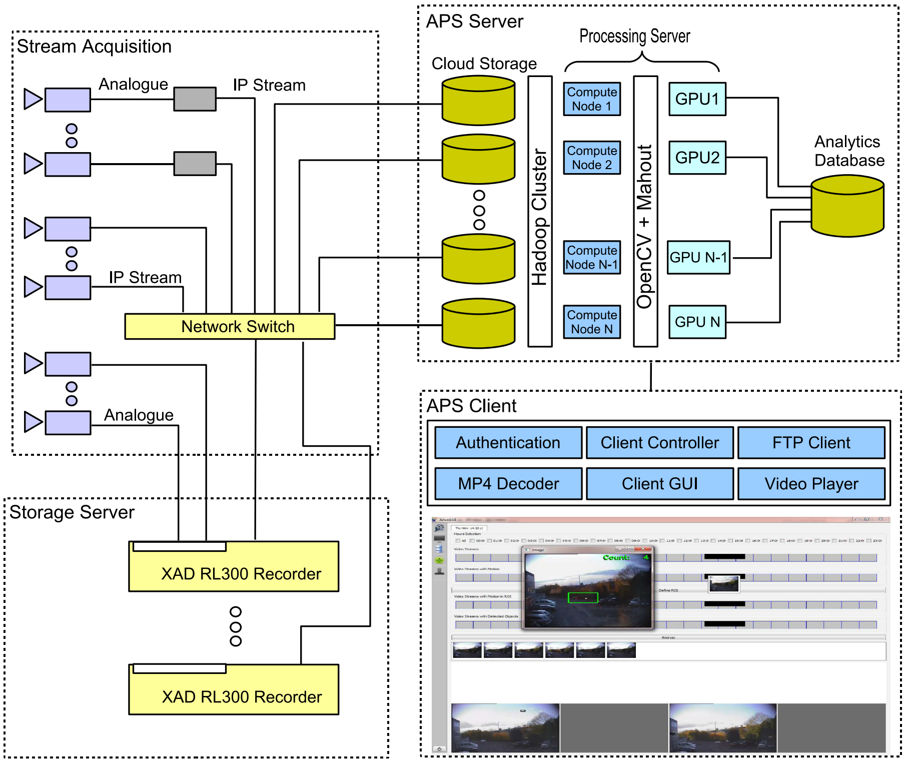
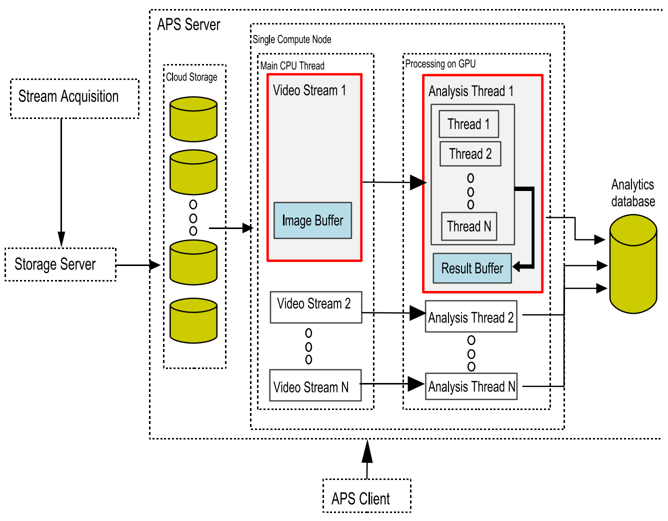
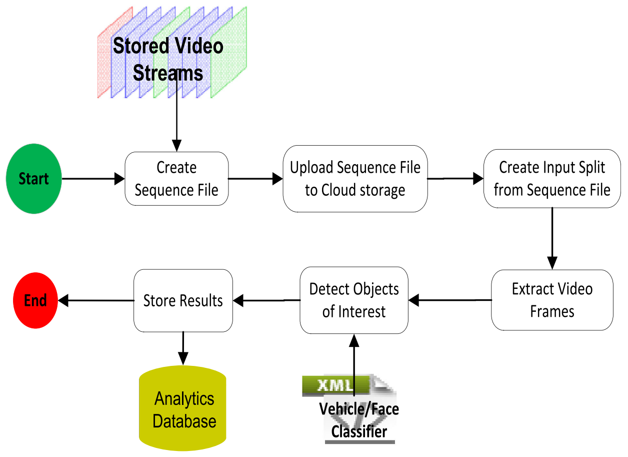


Fig. 1. System architecture of the video analysis framework.



(a) *Schematic Diagram*



(b) *Flow/Procedural Diagram*

Fig. 2. Video stream analysis on a compute node.

required during the video stream analysis. The video streams, specified in the analysis request, are fetched from the cloud storage. These video streams are analysed according to the analysis criteria and the analysis results are stored in the analytics database. The operator is notified of the completion of the analysis process. The operator can then access the analysis results from the database.

An "Analysis Request" comprises of the defined region of interest, an analysis criteria and the analysis time interval. The operator defines a region of interest in a video stream for an analysis. The analysis criteria defines parameters for detecting objects of interests (face, car, van or truck-vehicle) and size/colour based classification of the detected objects. The time interval represents the duration of analysis from the recorded video streams as the analysis of all the recorded video streams might not be required.

3.2 Framework Components

Our framework employs a modular approach in its design. At the top level, it is divided into client and server components (Fig. 1). The server component runs as a daemon on the cloud nodes and performs the main task of video stream analysis. Whereas, the client component supports multi-user environment and runs on the client machines (operators in our case). The control/data flow in the framework is divided into the following three stages:

- Video stream acquisition and storage
- Video stream analysis
- Storing analysis results and informing operators

The deployment of the client and server components is as follows: The Video Stream Acquisition is deployed at the video stream sources and is connected to the Storage Server through 1/10 Gbps LAN connection. The cloud based storage and processing servers are deployed collectively in the cloud based data center. The APS Client is deployed at the end-user sites. We explain the details of the framework components in the remainder of this section.

3.2.1 Video Stream Acquisition

The Video Stream Acquisition component captures video streams from the monitoring cameras and transmits to the requesting clients for relaying in a control room and/or for storing these video streams in the cloud data center. The captured video streams are encoded using H.264 encoder [22]. Encoded video streams are transmitted using Real Time Streaming Protocol (RTSP) [23] in conjunction with RTP/RTCP protocols [24]. The transmission of video streams is initiated on a client's request after authentication and establishing a session with the RTSP server.

The client is authenticated using CHAP protocol before transmitting video streams. The RTSP server sends a challenge message to the client in the CHAP protocol. The client responds to the RTSP server with a value obtained by a one-way hash function. The server compares this value with the value calculated with its own hash function. The RTSP server starts video stream transmission after a match of the hash values. The connection is terminated otherwise.

The video stream is transmitted continuously for one RTSP session. The video stream transmission stops only when the connection is dropped or the client disconnects itself. A new session will be established after a dropped connection and the client will be re-authenticated. The client is responsible for recording the transmitted video stream into video files and storing them in the cloud data center. Administrators in the framework are authorized to change quality of the captured video streams. Video streams are captured at 25 fps in the experimental results reported in this paper.

3.2.2 Storage Server

The scale and management of the data coming from hundreds or thousands of cameras will be in exabytes, let alone all of the more than four million cameras in UK. Therefore, storage of these video streams is a real challenge. To address this issue, H.264 encoded video streams received from the video sources, via video stream acquisition, are recorded as MP4 files on storage servers in the cloud. The storage server has RL300 recorders for real time recording of video streams. It stores video streams on disk drives and meta-data about the video streams is recorded in a database (Fig. 1). The received video streams are stored as 120 seconds long video files. These files can be stored in QCIF, CIF, 4CIF or in Full HD video format. These supported video

TABLE 2
Disk Space for One Month of Recorded Video Streams

Duration	Minimum Size	Maximum Size
2 Minutes (120 Seconds)	3 MB	120 MB
1 Hours (60 Minutes)	90 MB	3.6 GB
1 Day (24 Hours)	2.11 GB	86.4 GB
1 Week (168 Hours)	14.77 GB	604.8 GB
4 Weeks (672 Hours)	59.06 GB	2.419 TB

formats are explained in Section 2.3. The average size, frame rate, pixels per frame and the video resolution of each recorded video file for the supported video formats is summarized in Table 1.

The length of a video file plays an important role in the storage, transmission and analysis of a recorded video stream. The 120 seconds length of a video file is decided after considering the network bandwidth, performance and fault tolerance reasons in the presented framework. A smaller video file is transmitted quicker as compared to a large video file. Secondly, it is easier and quicker to re-transmit a smaller file after a failed transmission due to a network failure or any other reasons. Thirdly, the analysis results of a small video file are quickly available as compared to a large video file.

The average minimum and maximum size of a 120 seconds long video stream, from one monitoring camera, is 3 MB and 120 MB respectively. One month of continuous recording from one camera requires 59.06 GB and 2.419 TB of minimum and maximum disk storage respectively. The storage capacity required for storing these video streams of one month duration from one camera is summarized in Table 2.

3.2.3 Analytics Processing Server (APS)

The APS server sits at the core of our framework and performs the video stream analysis. It uses the cloud storage for retrieving the recorded video streams and implements a processing server as compute nodes in a Hadoop cluster in the cloud data center (Fig. 1). The analysis of the recorded video streams is performed on the compute nodes by applying the selected video analysis approach. The selection of a video analysis approach varies according to the intended video analysis purpose. The analytics results and meta-data about the video streams is stored in the Analytics Database.

Overall working of the framework is depicted in Fig. 1, the internal working of a compute node for analysing the video streams is depicted in Fig. 2a. An individual processing server starts analysing the video streams on receiving an analysis request from an operator. It fetches the recorded video streams from the cloud storage. The H.264 encoded video streams are decoded using the FFmpeg library and individual video frames are extracted. The analysis process is started on these frames by selecting features in an individual frame and matching these features with the features stored in the cascade classifier. The functionality of the classifier is explained in Section 4. The information about detected objects is stored in the analytics database. The user is notified after completion of the analysis process. Fig. 2b summarizes the procedure of analysing the stored video streams on a compute node.

3.2.4 APS Client

The APS Client is responsible for the end-user/operator interaction with the APS Server. The APS Client is deployed at the client sites such as police traffic control rooms or city council monitoring centers. It supports multi-user interaction and different users may initiate the analysis process for their specific requirements, such as object identification, object classification, or the region of interest analysis. These operators can select the duration of recorded video streams for analysis and can specify the analysis parameters. The analysis results are presented to the end-users after an analysis is completed. The analysed video streams along with the analysis results are accessible to the operator over 1/10 Gbps LAN connection from the cloud storage.

The APS Client is deployed at client sites such as police traffic control rooms or city council monitoring centers. A user from a client site connects to camera sources through video stream acquisition module. The video stream acquisition module transmits video streams over the network from camera sources. These video streams are viewed live or are stored for analysis. In this video stream acquisition/transmission model, neither changes are required in the existing camera deployments nor any additional hardware is needed.

3.2.5 Porting the Video Analytics Framework to a Public Cloud

The presented video analysis framework is evaluated on the private cloud at the University of Derby. Porting the framework to a public cloud such as Amazon EC2, Google Compute Engine or Microsoft Azure will be a straighforward process. We explain the steps/phases for porting the framework to Amazon EC2 in the remainder of this section.

The main phases in porting the framework to Amazon EC2 are data migration, application migration and performance optimization. The data migration phase involve uploading all the stored video streams from local cloud storage to the Amazon S3 cloud storage. The AWS SDK for Java will be used for uploading the video streams and AWS Management Console to verify the upload. The “Analytics Database” will be created in Amazon RDS for MySQL database. The APS is moved to Amazon EC2 in the application migration phase.

Custom Amazon Machine Images (AMIs) will be created on Amazon EC2 instances for hosting and running the APS components. These custom AMIs can be added or removed according to the varying workloads during the video stream analysis process. Determining the right EC2 instance size will be a challenging task in this migration. It is dependent on the processing workload, performance requirements and the desired concurrency for video streams analysis. The health and performance of the instances can be monitored and managed using AWS Management Console.

4 Video Analysis Approach

The video analysis approach detects objects of interest from the recorded video streams and classifies the detected objects according to their distinctive properties. The AdaBoost based cascade classifier [10] algorithm is applied for detecting objects from the recorded video streams. Whereas, size based classification is performed for the detected vehicles in the second case study.

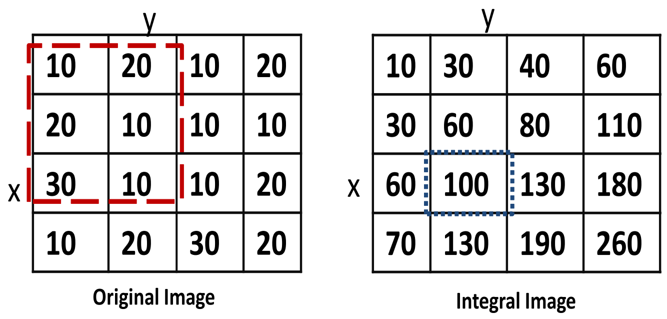


Fig. 3. Integral image representation.

In the AdaBoost based object detection algorithm, a cascade classifier combines weak classifiers into a strong classifier. The cascade classifier does not operate on individual image/video frame pixels. It rather uses integral image for detecting object features from the individual frames in a video stream. All the object features not representing an object of interest are discarded early in the object detection process. The cascade classifier based object detection increases detection accuracy, uses less computational resources and improves the overall detection performance. This algorithm is applied in two stages as explained below.

4.1 Training a Cascade Classifier

A cascade classifier combining a set of weak classifiers using real AdaBoost [12] algorithm is trained in multiple boosting stages. In the training process, a weak classifier learns about the object of interest by selecting a subset of rectangular features that efficiently distinguish both classes of the positive and negative images from the training data. This classifier is the first level of cascade classifier. Initially, equal weights are attached to each training example. The weights are raised for the training examples misclassified by the current weak classifier in each boosting stage. All of these weak classifiers determine the optimal threshold function such that mis-classification is minimized. The optimal threshold function is represented [12] as follow:

$$h_j(x) = \begin{cases} 1 & if \;\; p_j f_j(x) < p_j \theta_j \\ 0 & otherwise, \end{cases}$$

where x is the window, f_j is value of the rectangle feature, p_j is parity and θ_t is the threshold. A weak classifier with lowest weighted training error, on the training examples, is selected in each boosting stage. The final strong classifier is a linear combination of all the weak classifiers and has gone through all the boosting stages. The weight of each classifier in the final classifier is directly proportional to its accuracy.

The cascade classifier is developed in a hierarchical fashion and consists of cascade stages, trees, features and thresholds. Each stage is composed of a set of trees, trees are composed of features and features are composed of rectangles. The features are specified as rectangles with their x, y, height and width value and a tilted field. The threshold field specifies the branching threshold for a feature. The left branch is taken for the feature value less than the adjusted threshold and the right branch is taken otherwise. The tree values within a stage are accumulated and are compared with the stage threshold. The accumulated value is used to decide object detection at a stage. If this value is higher than the stage threshold the sub-windows are classified as an object and is passed to the next stage for further classification.

A cascade of classifiers increases detection performance and reduces computational power during the detection process. The cascade training process aims to build a cascade classifier with more features for achieving higher detection rates and a lower false positive rate. However, a cascade classifier with more features will require more computational power. The objective of the cascade classifier training process is to train a classifier with minimum number of features for achieving the expected detection rate and false positive rate. Furthermore, these features can encode ad hoc domain knowledge that is difficult to learn using finite quantity of the training data. We used opencv_harrtraining utility provided with OpenCV [25] to train the cascade classifier.

4.2 Detecting Objects of Interest from Video Streams Using Cascade Classifier

Object detection with a cascade classifier starts by scanning a video frame for all the rectangular features in it. This scanning starts from the top-left corner of the video frame and finishes at the bottom-right corner. All the identified rectangular features are evaluated against the cascade. Instead of evaluating all the pixels of a rectangular feature, the algorithm applies an integral image approach and calculates a pixel sum of all the pixels inside a rectangular feature by using only four corner values of the integral image as depicted in Fig. 3. The integral image results in faster feature evaluation than the pixel based evaluation. Scanning a video frame and constructing an integral image are computationally expensive tasks and always need optimization.

The integral image is used to calculate the value of the detected features. The identified features consist of small rectangular regions of white and shaded areas. These features are evaluated against all the stages of the cascade classifier. The value of any given feature is always the sum of the pixels within white rectangles subtracted from the sum of the pixels within shaded rectangles. The evaluated image regions are sorted out between objects and non-objects.

For reducing the processing time, each frame is scanned in two passes. A video frame is divided into sub-windows in the first pass. These sub-windows are evaluated against the first two stages of the cascade of classifiers (containing weak classifiers). A sub-window is not further evaluated, if it is eliminated in stage zero or evaluates to an object in stage one. The second pass evaluates only those sub-windows that are neither eliminated nor marked as objects in the first pass.

An attentional cascade is applied for reducing the detection time. In the attentional cascade, simple classifiers are applied earlier in the detection process and a candidate rectangular feature is rejected at any stage for a negative response from the classifier. The strong classifiers are applied later in the detection stages to reduce false positive detections.

It is important to mention that the use of a cascade classifier reduces the time and resources required in the object detection process. However, object detection by using a cascade of classifiers is still time and resource consuming. It can be further optimized by porting the parallel parts of the object detection process to GPUs.

It is also important to note that the machine learning based real AdaBoost algorithm for training the cascade classifier is not trained on the cloud. The cascade classifier is trained once on a single compute node. The trained cascade classifiers are used for detecting objects from the recorded video streams on the compute cloud.

4.3 Object Classification

The objects detected from the video streams can be classified according to their features. The colour, size, shape or a combination of these features can be used for the object classification.

In the second case study for vehicle detection, the vehicles detected from the video streams are classified into cars, vans or trucks according to their size. As explained above, the vehicles from the video streams are detected as they pass through the defined region of interest in the analysis request. Each detected vehicle is represented by a bounding box (a rectangle) that encloses the detected object. The height and the width of the bounding box is treated as the size of the detected vehicle.

All the vehicles are detected at the same point in video streams. Hence, the location of the detected vehicles in a video frame becomes irrelevant and the bounding box represents the size of the detected object. The detected vehicles are classified into cars, vans and trucks by profiling the size of their bounding boxes as follows. The vehicles with a bounding box of size less than (100×100) are classified as cars, (150×150) are classified as vans and all the detected vehicles above this size are classified as trucks. The object detection and classification results are explained in the experimental results section.

The faces detected from video streams, in the second case study, can be classified according to their gender or age. However, no classification is performed on the detected faces in this work. The object detection and classification results are explained in the experimental results section.

5 Experimental Setup

This section explains the implementation and experimental details for evaluating the video analysis framework. The results focus on the accuracy, performance and scalability of the presented framework. The experiments are executed in two configurations; cloud deployment and cloud deployment with Nvidia GPUs.

The cloud deployment evaluates scalability and robustness of the framework by analysing its different aspects including (i) video stream decoding time, (ii) video data transfer time to the cloud, (iii) video data analysis time on the cloud nodes and (iv) collecting the results after completion of the analysis.

The experiments on the cloud nodes with GPUs evaluate the accuracy and performance of the video analysis approach on state of the art compute nodes with two GPUs each. These experiments also evaluate the video stream decoding and video stream data transfer between CPU and GPU during the video stream analysis. The energy implications of the framework at different stages of the video analytics life cycle are also discussed towards the end of this section.

5.1 Compute Cloud

The framework is evaluated on the cloud resources available at the University of Derby. The cloud instance is running OpenStack Icehouse with Ubuntu LTS 14.04.1. It consist of six server machines with 12 cores each. Each server has two 6-core Intel Xeon processors running at 2.4 Ghz with 32 GB RAM and two Terabyte storage capacity. The cloud instance has a total of 72 processing cores with 192 GB of RAM and 12 TB of storage capacity. OpenStack Icehouse is providing a management and control layer for the cloud. It has a single dashboard for controlling the pool of computers, storage, network and other resources.

We configured a cluster of 15 nodes on the cloud instance and used it for evaluating the framework. Each of these 15 nodes have 100 GB of storage space, 8 GB RAM and a four VCPU running at 2.4 GHz. These experiments focus at different aspects of the framework such as analysis time of the framework, effect of task parallelism on each node, block size, block replication factor and number of compute/data nodes in the cloud. The purpose of these experiments is to test the performance, scalability and reliability of the framework with varying cloud configurations. The conclusions from these experiments can then be used for deploying the framework on a cloud with as many nodes as required by a user application.

The framework is executed on Hadoop MapReduce [21] for evaluations in the cloud. JavaCV, a Java wrapper of OpenCV is used as image/video processing library. Hadoop Yarn schedules the jobs and manages resources for the running processes on Hadoop. There is a NameNode for managing nodes and load balancing among the nodes, a DataNode/ComputeNode for storing and processing the data, a JobTracker for executing and tracking jobs. A DataNode stores video stream data as well as executes the video stream analysis tasks. This setting allows us to schedule analysis tasks on all the available nodes in parallel and on the nodes close to the data. Fig. 4 summarizes the flow of analysing video streams on the cloud.

5.2 Compute Node with GPUs

The detection accuracy and performance of the framework is evaluated on cloud nodes with two Nvidia GPUs. The compute nodes used in these experiments have Nvidia Tesla K20C and Nvidia Quadro 600 GPUs. Nvidia Tesla K20 has 5 GB DDR5 RAM, 208 GBytes/sec data transfer rate, 13 multiprocessor units and 2,496 processing cores. Nvidia Quadro 600 has 1 GB DDR3 RAM, 25.6 GBytes/sec data transfer rate, two multiprocessor units and 96 processing cores.

CUDA is used for implementing and executing the compute intensive parts of the object detection algorithm on a GPU. It is an SDK that uses Single Instruction Multiple Data (SIMD) parallel programming model. It provides fine-grained data parallelism and thread parallelism nested within coarse-grained data and task parallelism [26]. The CUDA program executing compute intensive parts of the object detection algorithm is called CUDA kernel. A CUDA program starts its execution on CPU, processes the data

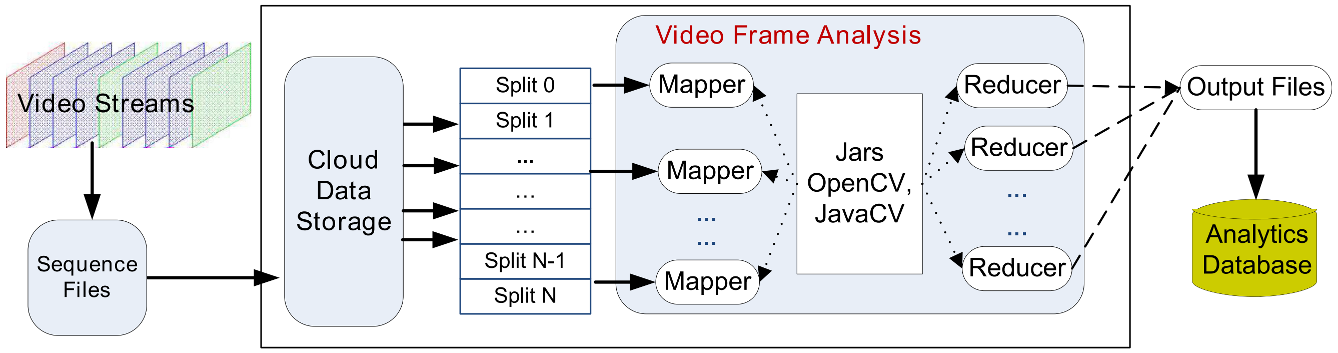


Fig. 4. Analysing video streams on the cloud.

with CUDA kernels on a GPU and transfers the results back to the host.

5.2.1 Challenges in Porting CPU Application to GPU

The main challenge in porting a CPU based application to a CUDA program is in identifying parts of the host application that can be executed in parallel and isolating the data required by these parts. After porting the parallel parts of the host application to CUDA kernels, the program and data are transferred to the GPU memory and the processed results are transferred back to the host with the CUDA API function calls.

The second challenge is faced while transferring the program data for kernel execution from CPU to GPU. This transfer is usually limited by the data transfer rates between CPU-GPU and the amount of available GPU memory.

The third challenge relates to the global memory access in a CUDA application. The global memory access on a GPU takes between 400 and 600 clock cycles as compared to two clock cycles of the GPU register memory access. The speed of memory access is also affected by the thread memory access pattern. The execution speed of a CUDA kernel will be considerably higher for coalesced memory access than that of non-coalesced memory access. The above challenges are taken into account while porting our CPU based object detection implementation to the GPU. The way, we tackled these challenges is detailed below.

5.2.2 What is Ported on GPU and Why

A video stream consists of individual video frames. All of these video frames are independent of each other from object detection perspective and can be processed in parallel. The Nvidia GPUs use SIMD model for executing CUDA kernels. Hence, video stream processing becomes an ideal application for porting to GPUs as the same processing logic is executed on every video frame.

We profiled the CPU based application and identified the compute intensive parts in it. Scanning a video frame, constructing an integral image, and deciding the feature detection are the compute intensive tasks and consumed most of the processing resources and time in the CPU based implementation. These functions are ported to GPU by writing CUDA kernels in our GPU implementation.

In the GPU implementation, the object detection process executes partially on CPU and partially on GPU. The CPU decodes a video stream and extracts video frames from it. These video frames and cascade classifier data are transferred to a GPU. The CUDA kernel processes a video frame and the object detection results are transferred back to the CPU.

We used OpenCV [25], an image/video processing library and its GPU component for implementing the analysis algorithms as explained in Section 4. JavaCV, a Java wrapper of OpenCV, is used in the MapReduce implementation. Some primitive image operations like converting image colour space, image thresholding and image masking are used from OpenCV library in addition to HaarCascade Classifier algorithm.

5.3 Image Datasets for Cascade Classifier Training

The UIUC image database [27] and FERET image database [28] are used for training the cascade classifier for detecting vehicles and faces from the recorded video streams respectively. The images in UIUC database are gray scaled and contain front, side and rear views of the cars. There are 550 single-scale car images and 500 non-car images in the training database. The training database contains two test image data sets. The first test set of 170 single-scale test images contains 200 cars at roughly the same scale as of the training images. Whereas, the second test set has 108 multi-scale test images containing 139 cars at various scales.

The FERET image database [28] is used for training and creating the cascade classifier for detecting faces. A total of 5,000 positive frontal face images and 3,000 non-face images are used for training the face cascade classifier. The classifier is trained for frontal faces only with an input image size of 20×20. It has 21 boosting stages.

The input images used for training both the cascade classifiers have a fixed size of 20×20. We used opencv_harrtraining utility provided with OpenCV for training both the cascade classifiers. These datasets are summarized in Table 3.

5.4 Energy Implications of the Framework

Energy consumed in a cloud based system is an important aspect of its performance evaluation. The following three

TABLE 3
Image Dataset with Cascade Classifier Parameters

	Image Size	Training Positive	Images Negative	Boosting Stages	Scale Factor
Vehicle	20×20	550	550	12	1.2
Face	20×20	5,000	3,000	21	1.2

are the major areas where energy savings can be made, leading to energy efficient video stream analysis.

1) Energy consumption on camera posts
2) Energy consumption in video stream transmission
3) Energy consumption during video stream analysis

5.4.1 *Energy Consumed on Camera Posts*

Modern state of the art cameras are employed which consume as little energy as possible. The solar powered, network enabled digital cameras are replacing the traditional power hungry cameras. These cameras behave like sensors and only activate themselves when an object appears in front of a camera, leading to considerable energy savings.

5.4.2 *Energy Consumed in Video Stream Transmission*

There are two ways we can reduce the energy consumption during the video stream transmission. The first one is by using compressions techniques on the camera sources. We are using H264 encoding in our framework which compresses the data up to 20 times before streaming it to a cloud data center. The second approach is by using an on source analysis to reduce the amount of data and thereby reducing the energy consumed in the streaming process. The on source analysis will have in-built rules on camera sources that will detect important information such as motion and will only stream the data that is necessary for analysis. We will employ a combination of these two approaches to considerably minimize the energy consumption in the video transmission process.

5.4.3 *Energy Consumption During Video Stream Analysis*

The energy consumed during the analysis of video data uses a large portion of energy consumption in the whole life cycle of a video stream. We have employed GPUs for efficient processing of the data. GPUs have lightweight processing cores than that of CPUs and consume less energy. Nvidia' s Fermi (Quadro 600) and Kepler's Tesla K20 GPUs are at least 10 times more energy efficient that the latest $\times 86$ based CPUs. We used both of these GPUs in the reported results.

The GPUs execute compute intensive part of the video analysis algorithms efficiently. We achieved a speed up of at least two times on cloud nodes with GPUs as compared to the cloud nodes without GPUs. In this way the energy use is reduced by half for the same amount of data. By ensuring the availability of video data closer to the cloud nodes also reduced the un-necessary transfer of the data during the analysis. We are also experimenting the use of in-memory analytics that will further reduce the time to analyse, leading to considerable energy savings.

6 Experimental Results

We present and discuss the results obtained from the two configurations detailed in Section 5. These results focus on evaluating the framework for object detection accuracy, performance and scalability of the framework. The execution of the framework on the cloud nodes with GPUs evaluates the performance and detection accuracy of the video analysis approach for object detection and classification. It also evaluates the performance of the framework for video stream decoding, video stream data transfer between CPU and GPU and the performance gains by porting the compute intensive parts of the algorithm to the GPUs.

The cloud deployment without GPUs evaluates the scalability and robustness of the framework by analysing different components of the framework such as video stream decoding, video data transfer from local storage to the cloud nodes, video data analysis on the cloud nodes, fault-tolerance and collecting the results after completion of the analysis. The object detection and classification results for vehicle/face detection and vehicle classification case studies are summarized towards the end of this section.

6.1 Performance of the Trained Cascade Classifiers

The performance of the trained cascade classifiers is evaluated for the two case studies presented in this paper i.e., vehicle and face detection. It is evaluated by the detection accuracy of the trained cascade classifiers and the time taken to detect the objects of interest from the recorded video streams. The training part of the real AdaBoost algorithm is not executed on the cloud resources. The cascade classifiers for vehicle and face detection are trained once on a single compute node and are used for detecting objects from the recorded video streams on the cloud resources.

6.1.1 *Cascade Classifier for Vehicle Detection*

The UIUC image database [27] is used for training the cascade classifier for vehicle detection. The details of this data set are explained in Section 5. Minimum detection rate was set to 0.999 and 0.5 was set as maximum false positive rate during the training. The test images data set varied in lightening conditions and background scheme.

The input images used in the classifier training has a fixed size of 20×20 for vehicles. Only those vehicles will be detected that have a similar size as of the training images. The recorded video streams have varying resolutions (Section 2.3) and capture objects at different scales. The objects of different sizes, than that of 20×20, from the recorded video streams can be detected by re-scaling the video frames. A scale factor of 1.1 means decreasing the video frame size by 10 percent. It increases the chance of matching size with the training model, however, re-scaling the image is computationally expensive and increases computation time during the object detection process.

However, the increased computation time of re-scaling process is compensated by the resulting re-scaled image. Setting this parameter to higher values of 1.2, 1.3 or 1.5 will process the video frames faster. However, the chances of missing some objects increase and the detection accuracy is reduced. The impact of increasing scaling factor on the supported video types is summarized in Table 4. It can be seen that with increasing scaling factor for the supported video formats, the number of candidate detection windows decreases, the object detection decreases and it takes more time to process a video frame with a higher scaling factor. The optimum scale factor value for the supported video formats was found to be 1.2 and was used in all of the experiments.

TABLE 4
Detection Rate with Varying Scaling Factor

Scale Factor	Single Scale Test Images				Multi-Scale Test Images			
	Detected Cars	Missed Cars	Detection Rate	Time (sec)	Detected Cars	Missed Cars	Detection Rate	Time (sec)
1.10	117	83	58.5%	1.01	76	63	54.68%	0.70
1.15	112	88	56%	0.82	71	68	51.08%	0.56
1.20	88	112	44%	0.77	56	83	40.29%	0.50
1.30	73	137	36.5%	0.69	46	93	33.09%	0.44
1.50	55	145	27.5%	0.58	37	102	26.62%	0.36

Another parameter that affects the quality of the detected objects during the object detection process is the minimum number of neighbors. It represents the number of neighbor candidate rectangles each candidate rectangle will have to retain in the detection process. A higher value decreases the number of false positives. In our experiments, 3$\sim$5 provided a balance of detection quality and processing time.

The compute time for the object detection process can be further decreased from domain knowledge. We know that any object less that 20 $\times$ 20 pixels is neither a vehicle nor a face in the video streams. Therefore, all the candidate rectangles that are smaller than 20 $\times$ 20 are rejected during the detection process and are not further processed.

Overall, performance of the trained classifier is 85 percent for the single-scale car images data set. Performance of the trained classifier was 77.7 percent for the mixed-scale car images data set. Since the classifier was trained with single-scale car images data set, relatively low performance of the classifier with multi-scale car images was expected. Best detection results were found with 12 boosting stages of the classifier. The object detection (vehicle) accuracy of the trained cascade classifier is summarized in Table 5.

6.1.2 Cascade Classifier for Face Detection

The procedure for training the cascade classifier for detecting faces from the image/video streams is the same as used for training the cascade classifier for vehicle detection. The cascade classifier for face detection is trained with FERET image database [28]. A value of 3 as minimum number of neighbors provided best detection. The minimum detection rate was set to 0.999 and 0.5 as the maximum false positive rate. The test image data set contained 752 frontal face images having varying lightening conditions and background schemes. The detection accuracy of the trained cascade classifier for frontal faces is 99.91 percent.

A higher detection rate of the frontal face classifier in comparison to the vehicle detection classifier is due to less variation in the frontal face poses. The classifier for vehicle detection is trained from the images containing frontal, side and rear poses of the vehicles. Whereas, the classifier for frontal face has information about the frontal face pose only.

TABLE 5
Trained Classifier Detection Accuracy

	# of Cars	Hit Rate	Miss Rate
Single-Scale Cars	200	85%	15%
Multi-Scale Cars	139	77.7%	22.3%
Faces	752	99.91%	0.09%

6.2 Video Stream Analysis on GPU

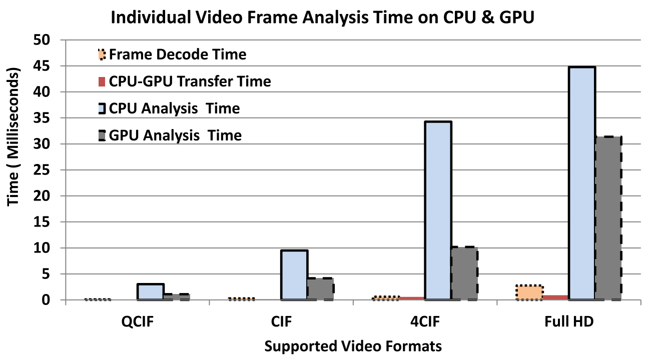
The video analytics framework is evaluated by analyzing the video streams for detecting objects of interest for two applications (vehicle/face). By video stream analysis, we mean decoding a video stream and detecting objects of interest, vehicle or face, from it. The results presented in this section focus on the functionality and computational performance required for analyzing the video streams. The analysis of a video stream on a GPU can be broken down into the following four steps: 1) Decoding a video stream, 2) Transferring a video frame from CPU to GPU memory, 3) Processing the video frame data, 4) Downloading the results from GPU to CPU.

The total video stream analysis time on a GPU includes video stream decoding time on the CPU, the data transfer from CPU to GPU, the processing time on GPU, and the results transfer time from GPU to CPU. Whereas, the total video stream analysis time on a CPU includes video stream decoding time and video stream processing time. It is important to note that no data transfer is required in the CPU implementation as the video frame data is being processed by the same CPU. The time taken on all of these steps for CPU and GPU executions is explained in the rest of this section.

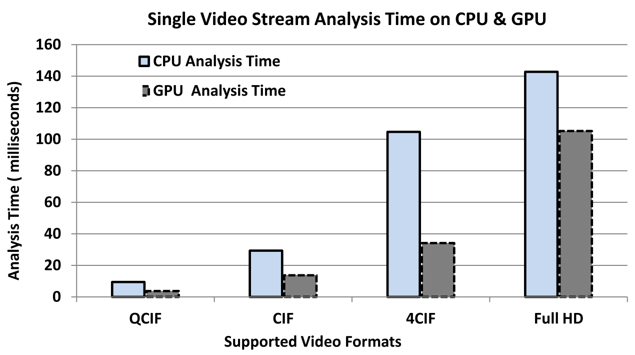
6.2.1 Decoding a Video Stream

Video stream analysis is started by decoding a video stream. The video stream decoding is performed using FFmpeg library and involves reading a video stream from the hard disk or from the cloud data storage and extracting video frames from it. It is an I/O bound process and can potentially make the whole video stream analysis very slow, if not handled properly. We performed a buffered video stream decoding to avoid any delays caused by the I/O process. The recorded video stream of 120 seconds duration is read into buffers for further processing. The buffered video stream decoding is also dependent on available amount of RAM on a compute node. The amount of memory used for buffered reading is a configurable parameter in our framework.

The video stream decoding time for extracting one video frame for the supported video formats varied from between 0.11 to 2.78 milliseconds. The total time for decoding a video stream of 120 seconds duration varied between 330 milliseconds to 8.34 seconds for the supported video formats. It can be observed from Fig. 5a that less time is taken to decode a lower resolution video format and more time to decode higher resolution video formats. The video stream decoding time is same for both CPU and GPU implementations as the video stream decoding is only done on CPU.



(a)



(b)

Fig. 5. (a) Frame decode, transfer and analysis time, (b) Total analysis time of one video stream on CPU & GPU.

6.2.2 Transfer Video Frame Data from CPU to GPU Memory

Video frame processing on GPU requires transfer of the video frame and other required data from CPU memory to the GPU memory. This transfer is limited by the data transfer rates between CPU and GPU and the amount of available memory on the GPU. The high end GPUs such as Nvidia Tesla and Nvidia Quadro provide better data transfer rates and have more available memory. Nvidia Tesla K20 has 5 GB DDR5 RAM and 208 GBytes/sec data transfer rate and Quadro 600 has 1GB DDR3 RAM and a data transfer rate of 25.6 GBytes/sec. Whereas, lower end consumer GPUs have limited on-board memory and are not suitable for video analytics.

The size of data transfer from CPU memory to GPU memory depends on the video format. The individual video frame data size for the supported video formats varies from 65 KB to 2.64 MB (the least for QCIF and the highest for the Full HD video format). The data transfer from CPU memory to the GPU memory took between 0.02 to 0.89 milliseconds for an individual video frame. The total transfer time from CPU to GPU for a QCIF video stream took only 60 milliseconds and 2.67 seconds for a Full HD video stream. Transferring the processed data back to CPU memory from GPU memory consumed almost the same time. The data size of a video frame for the supported video formats and the time taken to transfer this data from CPU to GPU is summarized in Table 6. An individual video frame reading time, the transfer time from CPU to GPU and the processing time on the CPU and on the GPU is graphically depicted in Fig. 5a. No data transfer is required for the CPU implementation as CPU processes a video frame directly from the CPU memory.

6.2.3 Processing Video Frame Data

The processing of data for object detection on a GPU is started after all the required data is transferred from CPU to GPU. The data processing on a GPU is dependent on the available CUDA processing cores and the number of simultaneous processing threads supported by a GPU. The processing of an individual video frame means processing all of its pixels for detecting objects from it using the cascade classifier algorithm. The processing time for an individual video frame of the supported video formats varied between 1.09 milliseconds to 30.38 milliseconds. The total processing time of a video stream on a GPU varied between 3.65 seconds to 105.14 seconds.

The total video stream analysis time on a GPU includes video stream decoding time, the data transfer time from CPU to GPU, the video stream processing time, and transferring the processed data back to the CPU memory from the GPU memory. The analysis time for a QCIF video stream, of 120 seconds duration, is 3.65 seconds and the analysis time for a Full HD video stream of the same duration is 105.14 seconds.

The processing of a video frame on a CPU does not involve any data transfer and is quite straightforward. The video frame data is already available in the CPU memory. The CPU reads the individual frame data and applies the algorithm on it. The CPU has less processing cores than that of a GPU and takes more time to process an individual video frame than on a GPU. It took 3.03 milliseconds for processing a QCIF video frame and 44.79 milliseconds for processing a Full HD video frame. The total processing time of a video stream for the supported video formats varied between 9.09 seconds to 134.37 seconds. Table 6 summarizes the individual video frame processing time for the supported video formats.

TABLE 6
Single Video Stream Analysis Time for Supported Video Formats

Video Format	Video Stream Resolution	Frame Decode Time	Frame Data Size	Frame Transfer Time (CPU-GPU)	Single Frame Analysis Time		Single Video Analysis Time	
					CPU	GPU	CPU	GPU
QCIF	177 × 144	0.11 msec	65 KB	0.02 msec	3.03 msec	1.09 msec	9.39 sec	3.65 sec
CIF	352 × 288	0.28 msec	273 KB	0.12 msec	9.49 msec	4.17 msec	29.31 sec	13.71 sec
4CIF	704 × 576	0.62 msec	1.06 MB	0.59 msec	34.28 msec	10.17 msec	104.69 sec	34.13 sec
Full HD	1,920 × 1,080	2.78 msec	2.64 MB	0.89 msec	44.79 msec	30.38 msec	142.71 sec	105.14 sec

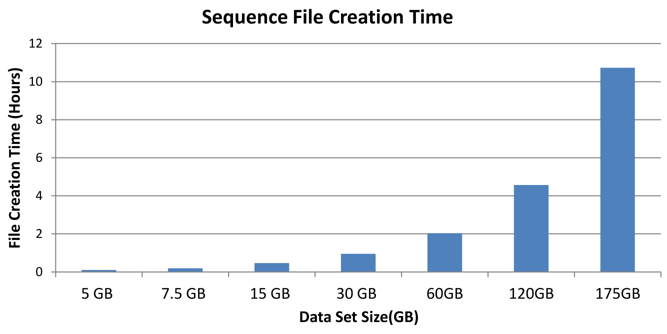


Fig. 6. Sequence file creation time with varying data sets.

The total video stream analysis time on CPU includes the video stream decoding and processing time. The total analysis time for a QCIF video stream is 9.39 seconds and the total analysis time for a Full HD video stream of 120 seconds duration is 142.71 seconds. It is obvious that the processing of a Full HD video stream on the CPU is slower and is taking more time than the length of a Full HD video stream. Each recorded video stream took 25 percent CPU processing power of the compute node. We were limited to analyse only three video streams in parallel on one CPU. The system was crashing with simultaneous analysis of more than three video streams.

In the GPU execution, we observe less speed up for QCIF and CIF video formats as compared to 4 CIF video format. QCIF and CIF are low resolution video formats and a part of the processing speed up gain is over-shadowed by the data transfer overhead from CPU memory to the GPU memory. The highest speed up of 3.07 times is observed for 4CIF video format and is least affected by the data transfer overhead, as can be observed in Fig. 5b.

6.3 Analysing Video Streams on Cloud

We explain the evaluation results of the framework on the cloud resources in this section. These results focus on evaluation of the scalability of the framework. The analysis of a video stream on the cloud using Hadoop [21] is evaluated in three distinct phases.

1) Transferring the recorded video stream data from storage server to the cloud nodes
2) Analysing the video stream data on the cloud nodes
3) Collecting results from the cloud nodes

Hadoop MapReduce framework is used for processing the video frames data in parallel on the cloud nodes. The input video stream data is transferred into the Hadoop file storage (HDFS). This video stream data is analysed for object detection and classification using the MapReduce framework. The meta-data produced is then collected and stored in the Analytics database for later use (Fig. 4).

Each cloud node executes one or more "analysis tasks". An analysis task is a combination of map and reduce tasks and is generated from the analysis request submitted by a user. A map task in our framework is used for processing the video frames for object detection, classification and generating analytics meta-data. The reduce task writes the meta-data back into the output sequence file. A MapReduce job splits input sequence file into independent data chunks and each data chunk becomes input to a map task. The output sequence file is downloaded and the results are stored in the Analytics database. It is important to mention that the MapReduce framework takes care of the scheduling map and reduce tasks, monitoring their execution and rescheduling the failed tasks.

6.3.1 *Creating Sequence File from Recorded Video Streams*

The H.264 encoded video streams, coming from camera sources, are first recorded in the storage server as 120 seconds long files (Section 3). The size of one month of the recorded video streams in 4 CIF format is 175 GB. Each video file has a frame rate of 25 frames per second. Therefore, each video file has 3000 (=120*25) individual video frames. The individual video frames are extracted as images and these images file are saved as PNG files before transferring them into the cloud data storage. The number of 120 seconds video files in one hour is 30 and in 24 hours is 720. The total number of image files for the 24 hours recorded video streams from one camera source is 216,000 (=720*3000).

These small image files are not suitable for directly processing with the MapReduce framework. As the MapReduce framework is designed for processing large files and processing a small file will decrease the overall performance. These files also require lots of disk seeks, inefficient data access pattern and increased hopping from DataNode to DataNode due to the distribution of the files across the nodes. These files are first converted into a large file which is suitable for processing with the MapReduce framework. The process of converting these small files into a large file and transferring to the cloud nodes is explained below.

These small files are stored as Hadoop sequence files. Hadoop sequence files use input image/video file name as a key and the contents of the file are recorded as the value in a sequence file. The header of a sequence file contains information on the key/value class names, version, file format, meta-data about the file and sync marker to denote the end of the header. The header is followed by the records which constitute the key/value pairs and their respective lengths. The sequence file uses compression. This results in consuming less disk space, less I/O and reduced network bandwidth usage. The cloud based analysis phase breaks these sequence file into input splits and operates on each input split independently.

The only drawback with the sequence file is that converting the existing data into sequence files is a slow and time consuming process. The recorded video streams are converted into sequence files by using a batch process at the source. The sequence files are then transferred to cloud data storage for object detection and classification.

6.3.2 *Sequence File Creation Time with Varying Data Sets*

We used multiple data sets that varied from 5 to 175 GB for generating these results. The 175 GB data set represents the recorded video streams, in 4CIF format, for one month from one camera source. These data sets helped in evaluating different aspect of the framework. These files are converted into one sequence file before transferring to the cloud data storage. The sequence file creation time varied between 6.15

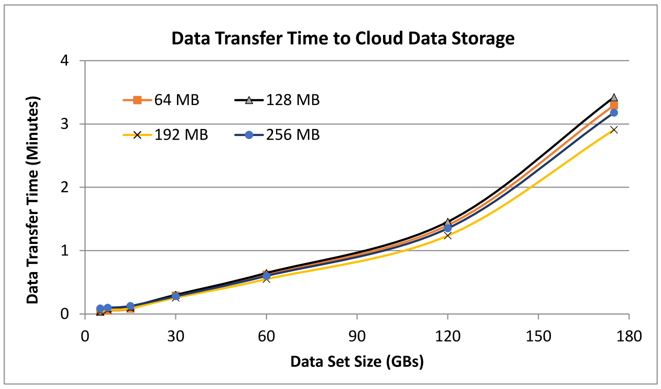


Fig. 7. Data transfer time to cloud data storage.

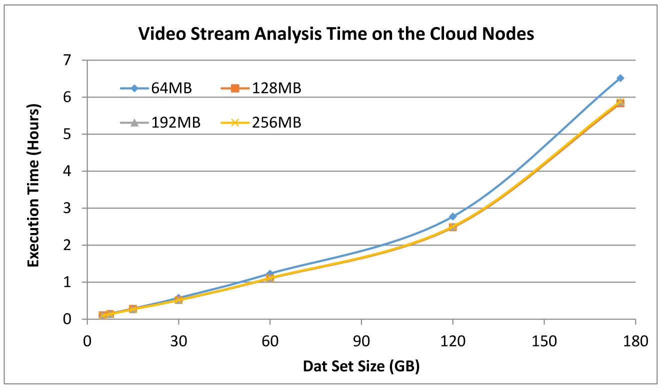


Fig. 8. Video stream analysis time on cloud nodes.

minutes to 10.73 hours for 5 to 175 GB data set respectively. Fig. 6 depicts the time needed to convert input data sets into a sequence file. The time needed to create a sequence file increases with the increasing size of the data set. However, this is a one off process and the resulting file remains stored in cloud data storage for all future analysis purposes.

6.3.3 Data Transfer Time to Cloud Data Storage

The sequence file is transferred to the cloud data storage for performing the analysis task of object detection and classification. The transfer time depends on the network bandwidth, data replication factor and the cloud data storage block size. For the data sets used (5 GB – 175 GB), this time varied from 2 minutes to 3.17 hours. The cloud storage block size varied from 64 to 256 MB and the data replication factor varied between 1 and 5. Fig. 7 shows data transfer time for varying block sizes. Varying block size does not seems to affect the data transfer time. Varying replication factor increases the data transfer time as each block of input data is replicated as many times as the replication factor in the cloud data storage. However, varying block size and varying replication factor do affect the analysis performance as explained below.

Analysing video streams on cloud nodes. The scalability and robustness of the framework is evaluated by analysing the recorded video streams on the cloud nodes. We explain the results of analysing the multiple data sets, varying from 5 to 175 GB, on the cloud nodes. We discuss the execution time with varying HDFS block sizes and the resources consumed during the analysis task execution on the cloud nodes. All of the 15 nodes in the cloud are used in these experiments. The performance of the framework on the cloud is measured by measuring the time it takes to analyse the data set of varying sizes, object detection performance and the resources consumed during the analysis task.

The block size is varied from 64 to 256 MB for observing the effect of changing block size on the Map task execution. With increasing data set size, an increasing trend is observed for the Map/reduce task execution time (Fig. 8). Varying block size has no major effect on the execution time and all the data sets consumed almost the same time for each block size. The execution time varied between 6.38 minutes and 5.83 hours for 5 and 175 GB data sets respectively. The analysis time for varying data sets and block sizes on the cloud is summarized in Table 7.

All the block sizes consumed almost the same amount of memory except the block size 64 MB. However, the 64 MB block size required more physical memory for completing the execution as compared to other block sizes (Fig. 10). The 64 MB block size is less than the default cloud storage block size (128 MB) and produces more data blocks to be processed by the cloud nodes. These smaller blocks cause management and memory overhead. The map tasks become inefficient with the smaller block sizes and more memory is needed to process these smaller block sizes. The system crashed when less memory is allocated to a container with 64 MB block size and required compute nodes with 16 GB of RAM. The total memory consumed by varying data sets is summarized in Fig. 10.

6.3.4 Robustness with Changing Cluster Size

The objective of this set of experiments is to measure the robustness of the framework. It is measured by total analysis time and the speedup achieved with varying number of cloud nodes. The number of cloud nodes is varied from 3 to 15 and the data set is varied from 5 to 175 GB.

We measured the time taken by one analysis task and the total time taken for analysing the whole data set with varying number of cloud nodes. Each analysis task takes a minimum time that cannot be reduced beyond a certain limit. The inter process communication, data read, and write from cloud storage are the limiting factors in reducing the execution time. However, the total analysis time decreases with the increasing number of nodes. It is summarized in Table 8.

The execution time for 175 GB data set with varying number of nodes for three different cloud storage block sizes is depicted in Fig. 9. The execution time shows a decreasing trend with increasing number of nodes. When the framework is executing with three nodes, it takes about 27.80 hours to analyse 175 GB data set. Whereas, the analysis of the same data set with 15 nodes is completed in 5.83 hours.

TABLE 7
Analysis Time for Varying Data Sets on the Cloud (Hours)

Block	Video Data Set Size (GBs)						
Size	5	7.5	15	30	60	120	175
64 MB	0.11	0.15	0.29	0.58	1.23	2.77	6.52
128 MB	0.10	0.14	0.28	0.52	1.10	2.48	5.83
256 MB	0.11	0.14	0.27	0.52	1.11	2.50	5.88

TABLE 8
Analysis Task Execution Time with Varying Cloud Nodes

Nodes	Tasks per Node	Execution Time Single Task (Seconds)	Execution Time All Tasks (Hours)
15	94	13.01	5.83
12	117	15.19	7.10
9	156	21.82	7.95
6	234	30.57	14.01
3	467	61	27.80

6.3.5 *Task Parallelism on Compute Nodes*

The total number of the analysis tasks is equal to the number of input splits. The number of analysis tasks running in parallel on a cloud node is dependent on the input data set, available physical resources and the cloud data storage block size. For the dataset size of 175 GB and with the default cloud storage block size of 128 MB, 1,400 map/reduce tasks are launched.

We varied the number of nodes from 3 to 15 in this set of experiments. The number of analysis tasks on each node increases with decreasing number of nodes. The increased number of tasks per node reduces performance of the whole framework. Each task has to wait longer to get scheduled and executed due to the over occupied physical resources on the compute nodes. The framework performance with varying number of nodes is summarized in Table 8.

The analysis time for 5 to 175 GB data sets with varying block sizes is summarized in Table 7 and is graphically depicted in Fig. 8. It is observed that analysis time of a data set with a larger block size is less as compared to a smaller block size. Less number of map tasks, with large block size, are better suited to process a data set as compared to a small block size. The reduced number of tasks reduces memory and management overhead on the compute nodes. However, input splits of 512 and 1,024 MB did not fit in the cloud nodes with 8 GB RAM and required compute nodes with large memory of 16 GB. The variation in the block size did not affect the execution time of the Map tasks. The 175 GB data set with 512 MB block size consumed the same time as of 256 MB or other block sizes. However, the larger block sizes required more time to transfer data and larger compute nodes to process the data.

Storing the results in analytics database. The reducers in map/reduce tasks write the analysis results to one output

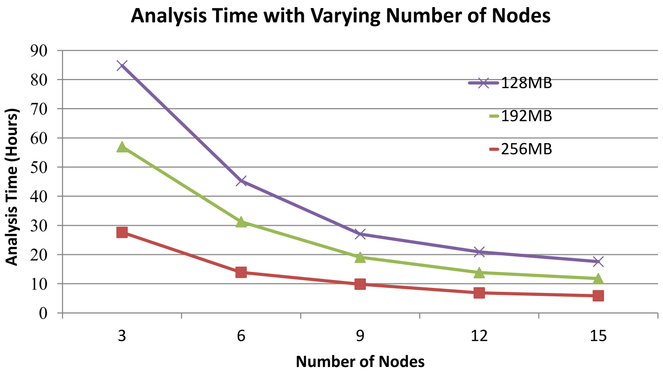


Fig. 9. Analysis time with varying number of the cloud nodes.

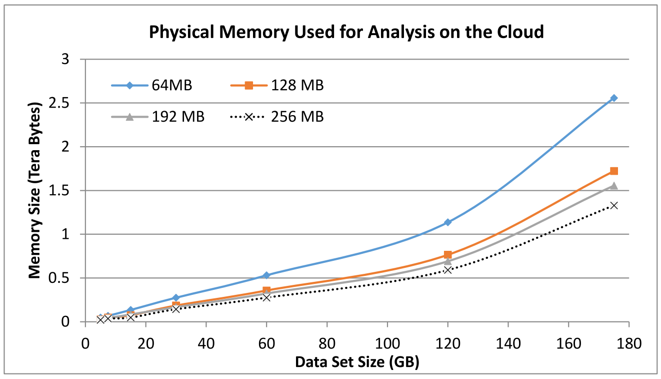


Fig. 10. Total memory consumed for analysis on the cloud.

sequence file. This output sequence file is processed separately, by a utility, for extracting analysis meta-date from it. The meta-data is stored in the Analytics database.

6.4 Object Detection and Classification

The object detection performance of the framework, its scalability and robustness is evaluated in the above set of experiments. This evaluation is important for the technical viability and acceptance of the framework. Another important evaluation aspect of the framework is the count of detected and classified objects after completing the video stream analysis.

The count of detected faces in the region of interest from the one month of video streams data is 294,061. In the second case study, total detected vehicles from the 175 GB data set are 160,954. These vehicles are moving in two different direction from the defined region of interest. The vehicles moving in are 46,371 and cars moving out are 114,574. The classification of the vehicles is based on their size. A total of 127,243 cars, 26,261 vans, and 7,441 trucks pass through the defined region of interest. These number of detected faces and vehicles are largely dependent on the input data set used for these case studies and may vary depending on the video stream duration.

7 Conclusions & Future Research Directions

The cloud based video analytics framework for automated object detection and classification is presented and evaluated in this paper. The framework automated the video stream analysis process by using a cascade classifier and laid the foundation for the experimentation of a wide variety of video analytics algorithms.

The video analytics framework is robust and can cope with varying number of nodes or increased volumes of data. The time to analyse one month of video data depicted a decreasing trend with the increasing number of nodes in the cloud, as summarized in Fig. 9. The analysis time of the recorded video streams decreased from 27.80 to 5.83 hours, when the number of nodes in the cloud varied from 3-15. The analysis time would further decrease when more nodes are added to the cloud.

The larger volumes of video streams required more time to perform object detection and classification. The analysis

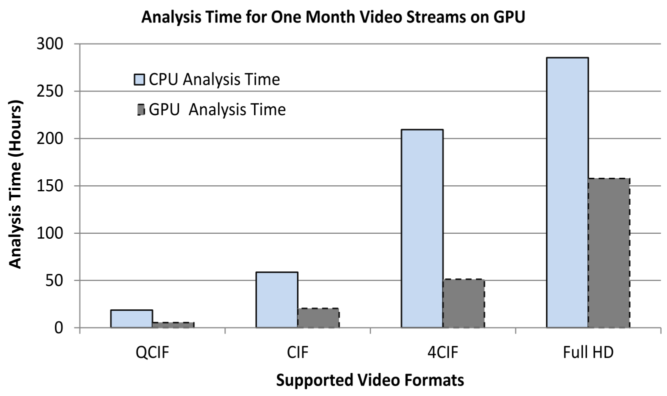


Fig. 11. Analysis time of one month of video streams on GPU.

time varied from 6.38 minutes to 5.83 hours, with the video stream data increasing from 5 to 175 GB.

The time taken to analyse one month of recorded video stream data on a cloud with GPUs is shown in Fig. 11. The speed up gain for the supported video formats varied according to the data transfer overheads. However, maximum speed up is observed for 4CIF video format. CIF and 4CIF video formats are mostly used for recording video streams from cameras.

A cloud node with two GPUs mounted on it took 51 hours to analyse one month of the recorded video streams in the 4 CIF format. Whereas, the analysis of the same data on the 15 node cloud took a maximum of 6.52 hours. The analysis of these video streams on 15 cloud nodes with GPUs took 3 hours. The cloud nodes with GPUs yield a speed up of 2.17 times as compared to the cloud nodes without GPUs.

In this work, we did not empirically measure the energy used in capturing, streaming and processing the video data. We did not measure the use of energy while acquiring video streams from camera posts and streaming them to the cloud data centre. This is generally perceived that GPUs are energy efficient and will save considerable energy by executing the video analytics algorithms. However, we did not empirically verify this fact in our experiments. This is one of the future directions of this work.

In future, we would also like to extend our framework for processing the live data coming directly from the camera sources. This data will be directly written into data pipeline by converting into sequence files. We would also extend our framework by making it more subjective. It will enable us to perform logical queries, such as , "How many cars of a specific colour passed yesterday" on video streams. More sophisticated queries like, "How many cars of a specific colour entered into the parking lot between 9 AM to 5 PM on a specific date" will also be included.

Instead of using sequence files, in future we would also like to use a NoSQL database such as HBase for achieving scalability in data writes. Furthermore, Hadoop fails to come up to the low latency requirements.

REFERENCES

[1] S. Ranford, "The picture in not clear: How many surveillance cameras are there in the UK?" British security industry association, 2015.
[2] D. Wood and K. Ball, "A report on the surveillance society," Sep. 2006, https://ico.org.uk/media/1042391/surveillance-society-summary-06.pdf
[3] M. Gill and A. Spriggs, *Assessing The Impact of CCTV*. London, U.K.: Home Office Research, Development and Statistics Directorate, Feb. 2005.
[4] S. J. McKenna and S. Gong, "Tracking colour objects using adaptive mixture models," *Image Vis. Comput.*, vol. 17, pp. 225–231, 1999.
[5] D. Koller, J. W. W. Haung, J. Malik, G. Ogasawara, B. Rao, and S. Russel, "Towards robust automatic traffic scene analysis in real-time," in *Proc. Int. Conf. Pattern Recog.*, 1994, pp. 126–131.
[6] J. S. Bae and T. L. Song, "Image tracking algorithm using template matching and PSNF-m," *Int. J. Control, Autom., Syst.*, vol. 6, no. 3, pp. 413–423, Jun. 2008.
[7] J. Hsieh, W. Hu, C. Chang, and Y. Chen, "Shadow elimination for effective moving object detection by gaussian shadow modeling," *Image Vis. Comput.*, vol. 21, no. 3, pp. 505–516, 2003.
[8] T. Abdullah, A. Anjum, M. Tariq, Y. Baltaci, and N. Antonopoulos, "Traffic monitoring using video analytics in clouds," in *Proc. 7th IEEE/ACM Intl. Conf. Utility Cloud Comput.*, 2014, pp. 39–48.
[9] C. Stauffer and W. E. L. Grimson, "Learning patterns of activity using real-time tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 747–757, Aug. 2000.
[10] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recog.*, 2001, pp. 511–518.
[11] Y. Lin, F. Lv, S. Zhu, M. Yang, T. Cour, K. Yu, L. Cao, and T. Huang, "Large-scale image classification: Fast feature extraction and svm training," in *Proc. Conf. Comput. Vis. Pattern Recog.*, 2011, pp. 1689–1696.
[12] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," in *Proc. 11th Annu. Conf. Comput. Learning Theory*, 1998, pp. 80–91.
[13] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol.*, 2010, pp. 1–10.
[14] A. Ishii and T. Suzumura, "Elastic stream computing with clouds," in *Proc. 4th IEEE Int. Conf. Cloud Comput.*, 2011, pp. 195–202.
[15] Y. Wu, C. Wu, B. Li, X. Qiu, and F. Lau, "CloudMedia: When cloud on demand meets video on demand," in *Proc. 31st Int. Conf. Distrib. Comput. Syst.*, 2011, pp. 268–277.
[16] Vi-system, (2016). [Online]. Available: http://www.agentvi.com/
[17] Project BESAFE. (2016). [Online]. Available: http://www.besafe-project.net/
[18] "IVA 5.60 intelligent video analysis," Bosch, Tech. Rep., 2014.
[19] (2015). [Online]. Available: http://www.eptascape.com/products/eptaCloud.html
[20] K.-Y. Liu, T. Zhang, and L. Wang, "A new parallel video understanding and retrieval system," in *Proc. IEEE Int. Conf. Multimedia Expo*, 2010, pp. 679–684.
[21] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[22] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560–576, Jul. 2003.
[23] H. Schulzrinne, A. Rao, and R. Lanphier, "Real time streaming protocol (RTSP)," Internet Engineering Task Force (IETF), Apr. 1996, http://www.ietf.org/rfc/rfc2326.txt
[24] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," Internet Engineering Task Force (IETF), Apr. 1996, http://www.ietf.org/rfc/rfc2326.txt
[25] OpenCV. (2016). [Online]. Available: http://opencv.org/
[26] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, 1st ed., Reading, MA, US: Addison-Wesley, 2010.
[27] (2016). [Online]. Available: http://cogcomp.cs.illinois.edu/Data/Car
[28] (2016). [Online]. Available: http://www.itl.nist.gov/iad/humanid/feret/feret_master.html

Ashiq Anjum is a professor of distributed systems at the University of Derby, United Kingdom. His research interests include Data Intensive Distributed Systems, Parallel Computing, and High Performance Analytics platforms. He is currently investigating high performance distributed platforms to efficiently process video and genomics data.

Tariq Abdullah received the PhD degree from the TUDelft. He is a research engineer at the University of Derby, United Kingdom. His research interests include video analytics, big data analysis, machine learning, and distributed systems.

M. Fahim Tariq received the PhD degree from the University of Bristol. He is a director at the XAD Communications. His research interests are algorithm design, embedded systems, and video analytics.

Yusuf Baltaci received the PhD degree from the University of Bristol. He is a director at the XAD Communications. His research interests are hardware design, embedded systems, and video analytics.

Nick Antonopoulos received the PhD degree from the University of Surry and has more than 20 years of research and industry experience. He is a professor at the University of Derby. He has authored more than 80 publications in peer reviewed journals and conferences.