Traffic Monitoring Using Video Analytics in Clouds

Tariq Abdullah[1,2], Ashiq Anjum[1], M Fahim Tariq[2], Yusuf Baltaci [2], Nikos Antonopoulos[1]
[1]College of Engineering and Computing,
University of Derby, United Kingdom
[2]XAD Communications, Bristol, United Kingdom
[1]{t.abdullah, a.anjum, n.antonopoulos}@derby.ac.uk,
[2]{m.f.tariq, yusuf.baltaci}@xadco.com

***Abstract*—Traffic monitoring is a challenging task on crowded roads. Traditional traffic monitoring procedures are manual, expensive, time consuming and involve human operators. They are subjective due to the very involvement of human factor and sometimes provide inaccurate/incomplete monitoring results. Large scale storage and analysis of video streams were not possible due to limited availability of storage and compute resources in the past. Recent advances in data storage, processing and communications have made it possible to store and process huge volumes of video data and develop applications that are neither subjective nor limited in feature sets. It is now possible to implement object detection and tracking, behavioural analysis of traffic patterns, number plate recognition and automate security and surveillance on video streams produced by traffic monitoring and surveillance cameras.**

In this paper, we present a video stream acquisition, processing and analytics framework in the clouds to address some of the traffic monitoring challenges mentioned above. This framework provides an end-to-end solution for video stream capture, storage and analysis using a cloud based GPU cluster. The framework empowers traffic control room operators by automating the process of vehicle identification and finding events of interest from the recorded video streams. An operator only specifies the analysis criteria and the duration of video streams to analyse. The video streams are then automatically fetched from the cloud storage, decoded and analysed on a Hadoop based GPU cluster without operator intervention in our framework. It reduces the latencies in video analysis process by porting its compute intensive parts to the GPU cluster. The framework is evaluated with one month of recorded video streams data on a cloud based GPU cluster. The results show a speedup of 14 times on a GPU and 4 times on a CPU when compared with one human operator analysing the same amount of video streams data.

I. Introduction

Traffic monitoring in crowded traffic areas and on motorways is a challenging task. Manual registration of vehicles, counting vehicles using magnetic loops, and roadside traffic cameras are most commonly employed methods used by traffic police and transportation planning authorities. These approaches are subjective, inaccurate and at times may provide incomplete monitoring results. There is also a lack of classification and tracking of moving vehicles. These approaches do not automatically produce color, size and vehicle type information which are the key questions asked by law enforcement and emergency services while dealing with an incident. Moreover, these approaches are costly and time consuming to such an extent that their usefulness is sometimes questionable.

There are approximately 4 million to 5.9 million cameras in UK [1]. Camera based traffic monitoring and enforcement of speed restrictions have increased from just over 300,000 in 1996 to over 2 million in 2004 [2]. In the traditional traffic monitoring approaches, a video stream coming traffic monitoring cameras is viewed live in traffic control rooms or is recorded on a bank of DVRs or computer HDD for later processing. Depending upon the needs, the recorded video is retrospectively analyzed by police or other legal authorities. Manual analysis of video footage 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; making it impractical to go through recorded videos in a time constrained scenario. In an under staffed control room, an operator has to juggle between viewing live and recorded video contents while searching for an object of interest making the situation a lot worse [3].

The purpose of this research is to build a robust and high throughput cloud computing based solution for automatic analysis of video streams coming from traffic monitoring cameras and recorded in a cloud based storage. The term video analytics refers to the processing and analysis of video streams using computing resources. An operator, sitting in a traffic control room, only specifies the analysis criteria (explained in Section III) and the duration of video streams to analyse in the presented framework. The recorded video streams are then automatically fetched from the cloud storage, decoded and analysed on a Hadoop based GPU cluster without operator intervention. The operator is notified after completion of the analysis process and can access the analysis results from the cloud storage. The framework reduces latencies in video analysis process by using Nvidia GPUs. The cloud based solution offers the capability to analyse video streams for on-demand and on-the-fly monitoring and analysis of events.

The rest of the paper is organized as follow: The related work and state of the art is described in Section II. The presented video analytics framework is explained in Section III. This section also explains different components of our framework and their interaction. The algorithm used for detecting vehicles from the recorded video streams is explained in Section IV. Section V explains the experimental setup and discusses the results in great detail. Whereas, the paper is concluded in Section VI with some future research directions.

II. 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. We will conclude this section with salient features of our presented framework that are likely to bridge the gaps in existing research.

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 [4], background separation using Gaussian Mixture Models (GMM) [5], [6], [7] and cascade classifiers [8]. 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 [5], [6]. 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 analytics is proposed in [6]. Background frame differencing [9] is a variation of background separation approaches and identifies moving objects from their background in a video stream. It uses averaging and selective update methods [9] for updating the background in response to environmental changes. Background separation and frame differencing methods are not suitable in our case as these are computationally expensive.

A cascade of classifiers (termed as HaarCascade Classifier) [8] is an object detection approach and uses real AdaBoost [10] 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 cascade of classifiers for detecting vehicles in video streams for the results reported in Section V. The implementation details of this algorithm and steps of creating a cascade classifier for detecting vehicles is provided in Section IV.

Video Analytics in the Clouds

Large systems usually consist of hundreds or even thousands number of cameras covering over a wide area, streams are captured, 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 computational approaches can be a good choice for obtaining processing throughput. Hence, video stream processing in the clouds has recently become an active area of research to provide high computation, precision and efficiency to real time implementation of video traffic monitoring. However, major research focus has been on efficient video content retrieval using Hadoop [11], encoding/decoding [12], distribution of video streams [13] and on load balancing of computing resources for on-demand video streaming systems using cloud computing platforms [13], [14].

Video analytics have mainly been the focus of commercial vendors. Vi-System [15] offers an intelligent surveillance system with real time monitoring, tracking of an object within a crowd using analytical rules and provides alerts for different users on defined parameters. Vi-System does not work for recorded videos, analytic rules are limited and need to be defined in advance. SmartCCTV [16] provides optical based survey solutions, video incident detection systems, high end digital CCTV and is mainly used in UK transportation system. Project BESAFE [17] aimed for automatic surveillance of people, tracking their abnormal behaviour and detection of their activities using trajectories approach for distinguishing state of the objects. The main limitation of SmartCCTV and Project BESAFE is lack of scalability to a large number of streams and a requirement of 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. It can detect idle and removed objects as well as loitering, multiple line crossing, and trajectories of an object. EptaCloud [19] extends the functionality provided by IVA 5.60 and implements the system in a scalable environment. Intelligent Vision [20] is a tool for performing intelligent video analysis and for fully automated video monitoring of a premises with a rich set of features. IVA 5.60 and Intelligent Vision are not scalable and do 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. [21] proposed a hybrid parallel computing framework based on MapReduce programming model which supports multi-core and GPU architecture and the results suggest that such a model will be hugely beneficial for video processing and real time surveillance systems. We aim to use a similar approach in this research.

Existing cloud based video analytics approaches do not support recorded video streams [15], lack scalability [16], [17], and multi-core with GPU based approaches are still experimental [21]. IVA 5.60 [18] and Intelligent Vision [20] are not scalable, otherwise their approaches are close to the approach presented in this research.

The presented framework uses a cloud based storage to capture and record video streams and a GPU cluster to analyse the recorded video streams using cascade classifier object detection algorithm. This framework is explained in Section III and the video analysis algorithm used for detecting vehicles from the recorded video streams is detailed in Section IV.

III. Video Analytics Framework

The presented video analytics framework provides an end-to-end solution for video stream capture, storage, retrieval and

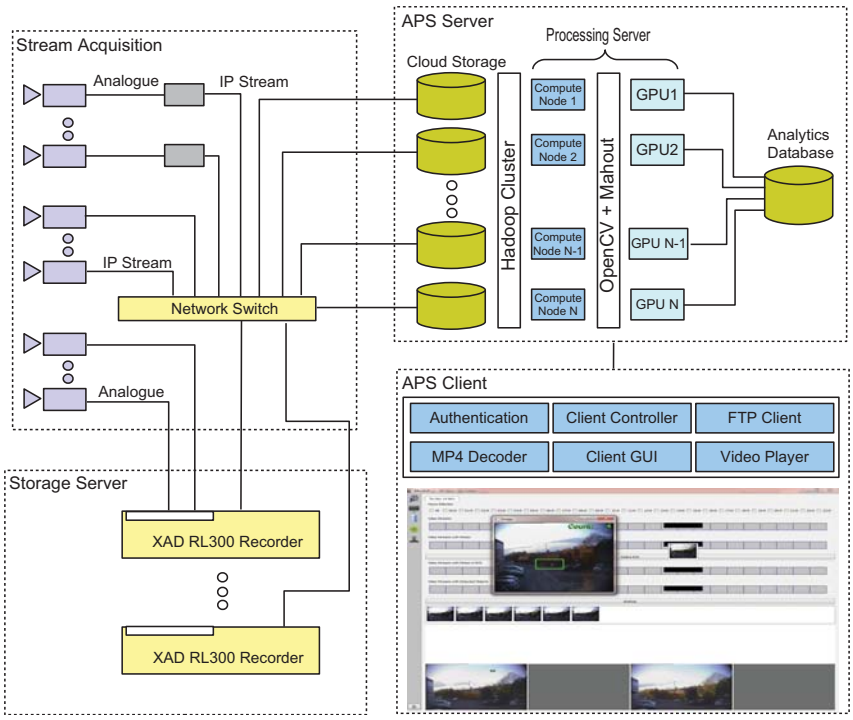

Figure 1: *System Architecture of the Stream Cloud*

processing. This framework makes the video stream analysis process efficient and reduces the processing latencies by porting its compute intensive parts to Nvidia GPUs**.** It empowers traffic control room operators by automating the process of identifying vehicles and finding events of interest.

This section outlines the presented framework, its different components and the interaction between them (Figure 1). The framework integrates both proprietary solutions for video storage and retrieval and the non-proprietary file servers (containing video streams) for video analytics. Video streams are captured and stored in a local cloud based storage from cameras installed on roads/motorways for traffic monitoring.

The analysis of these video streams is initiated on user's request (from the APS Client component as shown in Figure 1). A user defines region of interest in a video stream for analysis and selects an analysis criteria. The analysis criteria defines parameters for identification of different types of vehicles (car, van or truck) and color based classification of the identified vehicles. A user can specify the time interval for analysis from the recorded video streams as the analysis of all the video streams might not be required by a user. A defined region of interest, an analysis criteria and the analysis time interval are referred to as an Analysis Request in the rest of the paper.

An analysis request is sent to the Hadoop based GPU compute cluster for analysis. The compute cluster downloads the video stream, specified in the analysis request, from the cloud storage and performs analysis on the video stream. Analysis results are stored in a MySQL database and the user is notified of the completion of the analysis process. The user can then access the analysis results from the database.

Our framework employs a modular approach in its design. At the top level, it is divided into client and server components. The server component runs as a daemon on the server machines and performs the main task of video stream analysis using compute resources from the GPU compute cluster established locally. Whereas, the client component supports multi-user environment and runs on the client machines (traffic control rooms in our case). The control/data flow in the framework is divided into the following three stages:

- Video stream acquisition and storage
- Video stream processing for analysis
- Storing analysis results and informing end-users

The deployment of the client and server components is as follows: The 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 Server and the Processing Server are deployed collectively in a Hadoop based GPU cluster and all these components run as the Stream Cloud. The APS Client is deployed at the end-user sites. We explain the components of our framework in the remainder of this section.

A. Stream Acquisition

The Stream Acquisition component captures video streams from the traffic monitoring cameras and transmits to the requesting clients for relaying in traffic control room and/or for storing these video streams. The captured streams are encoded using H.264 encoder. Encoded video streams are transmitted using RTSP protocol [22] in conjunction with

Video Format	Frame Rate	Pixels per Frame	Video Resolution	Average Recorded Video Size
CIF (Common Intermediate Format)	29.97	99.0k	352 X 288	7.50MB
QCIF (Quarter CIF)	29.97	24.8k	176 X 144	2.95MB
4CIF	29.97	396k	704 X 576	8.30MB
Full HD (Full High Definition)	29.97	1.98M	1920 X 1080	9.35MB

Table I: *Supported Video Recording Formats*

RTP/RTCP protocols [23]. Transmission of video streams is initiated on a user's request. A user connects to the stream acquisition component by establishing an RTSP session. The user is authenticated using CHAP protocol before establishing a connection for stream transmission. The video stream delivery starts immediately after a client is authenticated and a session is established. 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. An explanation of important H.264 encoder parameters, selected values of these parameters, and the supported video formats used in the stream acquisition is provided below.

H.264 Encoder Parameters: H.264 is a block-oriented, motion compensation based video encoding format with lossy compression for the recording, compression, and distribution of video content from video stream sources [24]. The resolution mode of the encoder is set for constant quality with a constant bit rate of 200 kbps. Inter frame prediction modes point to the position of matching macroblocks in a reference frame and help in computing the motion vectors from encoded frames. The encoder supports 16x16, 8x16, 16x8 and 8x8 macroblocks for inter frame prediction modes. The intra prediction mode is set to 16x16 for less distortion in a video frame and for smoother encoding of a video frame. The encoder supports DC intra prediction mode only which is a mean of the upper and left-hand samples. The constant quantizer or QPmode is set to 10 for less toleration to the data loss during encoding of a video frame. The motion estimation resolution of the encoder is set to the integer pixel for reducing duplication (redundant data) among the adjacent frames. The search range of motion estimation is set to 32 for faster search and its search method is selected as the diamond search (DAI) for higher encoding speed. The intra blocks in P-frames is set to check high average boundary error (ABE) and applies an intra refresh rate of 300. The motion vectors are calculated for each video frame and are recorded in MP4 files (Section III-B).

Supported Video Formats: CIF, QCIF, 4CIF and Full HD video formats are supported for video stream recording in the presented framework. The resolution (number of pixels present in one frame) of a video stream in CIF format is 352x288 and each video frame has 99k pixels approximately. QCIF (Quarter CIF) is a low resolution video format and is used in setups with limited network bandwidth. Video stream resolution in QCIF format is 176x144 and each video frame has 24.8k pixels approximately. 4CIF video format has 4 times higher resolution (704x576) than that of CIF format and captures more details in each video frame. CIF and 4CIF formats are mostly used for acquiring video streams from the camera sources for traffic monitoring in our framework. Full HD (Full High Definition) video format captures video streams with 1920x1080 resolution and contains 24 times more details in a video stream than CIF format. It is used for high resolution video recording with availability of abundant disk storage and high speed internet connection such as fibre optic connection.

A higher resolution video stream presents a clearer image of the scene and captures more details. However, it also requires more network bandwidth to transmit the video stream and occupies more disk storage.

Other factors that may affect the video stream quality are *video bit rate* and *frames per second* (fps). Video bitrate represents the number of bits transmitted from a video stream source to the destination over a set period of time and is a combination of the video stream itself and mate-data about the video stream. Frames per second represents the number of video frames stuffed in a video stream in one second and determines the smoothness of a video stream. The video streams are captured with a constant bitrate of 200kbps and at 25 fps in the results reported in this paper. Table I summarizes different aspects such as frame rate, pixels per frame, video resolution and average recorded video size of the supported video formats.

B. Storage Server

H.264 encoded video streams received from the video sources, via stream acquisition, are recorded as MP4 files on the Storage Server. The storage server has proprietary XAD RL300 recorders. It stores video streams on disk drives and meta-data about the video streams is recorded in a database (see Figure 1). The acquired video streams are stored as 120 seconds long video files. This length is decided considering network bandwidth, performance and fault tolerance considerations in the presented framework. The average size of each file for the supported video formats is given in Table I. The motion vectors for each frame are recorded in a separate container of the MP4 file at the storage time.

The average minimum and maximum size of a 120 seconds long video stream, from one traffic monitoring camera, is 3MB

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

Table II: *Average Disk Space Requirements for One Month of the Recorded Video Streams*

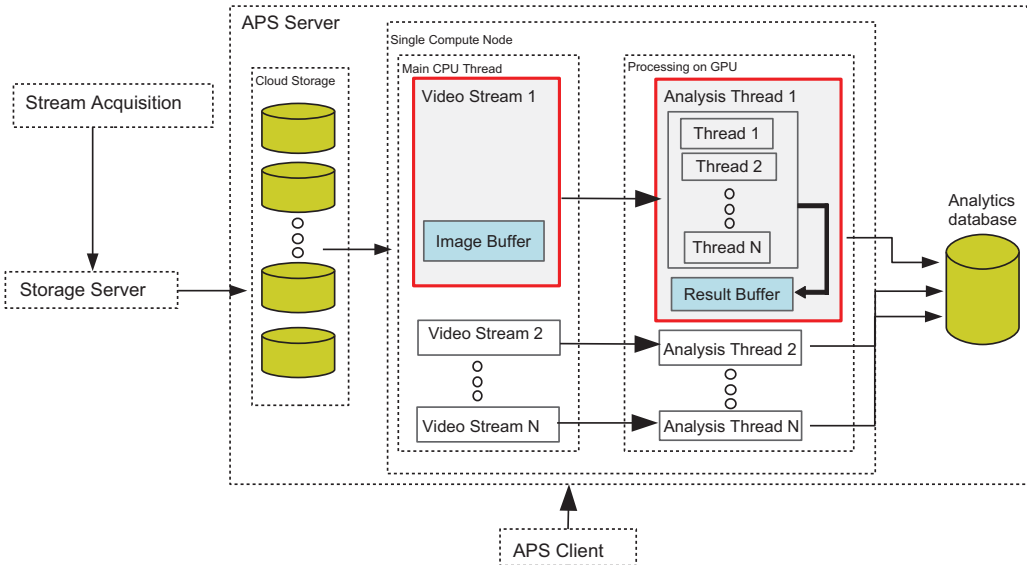

Figure 2: *Stream Processing in a Compute Node*

and 180MB respectively. One month of continuous recording from one camera requires 59.06GB and 2.419TB of minimum and maximum disk storage respectively. The storage capacity required for storing these video streams from one camera is summarized in Table II. The scale and management of the data coming from hundreds or thousands of cameras is in Exabytes, let alone all of the more than 4 million cameras in UK.

C. APS Server

The APS server sits at the core of our framework and performs the video stream analysis. It consists of a cloud storage for storing the recorded video streams, a processing server having compute nodes with Nvidia GPUs in a Hadoop cluster and the Cloud Storage (as shown in Figure 1). The analysis of the recorded video streams is performed on the compute nodes by applying the cascade classifier algorithm (see Section IV for details). Selection of an algorithm varies according to the intended purpose of the analysis. The analytics results and meta-data about the video streams is stored in the Analytics Database.

The processing server starts analysing the video streams on receiving the analysis request from an end-user. It downloads 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 (Section IV). The information about detected vehicles is stored in the analytics database. The user is notified after completion of the analysis process. Overall working of a compute node, for processing a video stream, in the processing server is depicted in Figure 2.

D. APS Client

The APS Client is responsible for the end-user interaction with the APS Server for traffic monitoring and supports multi-user interaction. Different users may initiate the analysis process for their specific requirements, such as vehicle identification, vehicle classification, or the region of interest analysis. These users can select the duration of recorded video streams for analysis and can specify the analysis parameters (vehicle identification, tracking etc). 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 end-users over 1/10 Gbps LAN connection from the cloud storage.

IV. Video Analysis Algorithm

We used real AdaBoost based cascade classifier [8] algorithm for detecting vehicles from the video streams. This algorithm is applied in two stages. First, a cascade classifier is trained from the training image data set. The trained classifier is then used for detecting vehicles from the recorded video streams.

Creating a Cascade Classifier: A cascade classifier combining a set of weak classifiers using real AdaBoost [10] algorithm is trained in multiple boosting stages. In the training process, a weak classifier learns about the vehicles 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 mathematically represented 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_t 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 classifier and has gone through all the boosting stages. The weight of each classifier in the final classifier is directly proportional to its accuracy.

Building a cascade of classifiers increases detection performance and reduces computation power during the detection process. The cascade training process aims to build a cascade classifier with more features for achieving higher detection rate 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 a utility provided with OpenCV [25] to train the cascade classifier for detecting vehicles from the recorded video streams. The parameters used in the cascade classifier training process and its performance results are detailed in Section V.

Detecting vehicles from video streams using cascade classifier: The algorithm [8] starts scanning an individual video frame for identifying rectangular features from the top-left corner and finishes at the bottom-right corner. All the identified rectangular features are evaluated against the cascade classifier in the detection process. We are not interested in the smaller rectangular features while detecting vehicles from video streams and discarded all the rectangular features that are less than 20x20 pixels.

Instead of evaluating all the pixels of a rectangular regions, the algorithm applies an integral image approach and calculates a pixel sum of all the pixels inside a rectangular feature by using only 4 corner values of the integral image as depicted in Figure 3. The integral image results in faster feature evaluation than the pixel based systems. Scanning a video frame and constructing an integral image are computationally expensive tasks and can be further optimized as explained in SectionV-B.

The identified features consist of small rectangular regions of white and shaded areas and are evaluated against all the

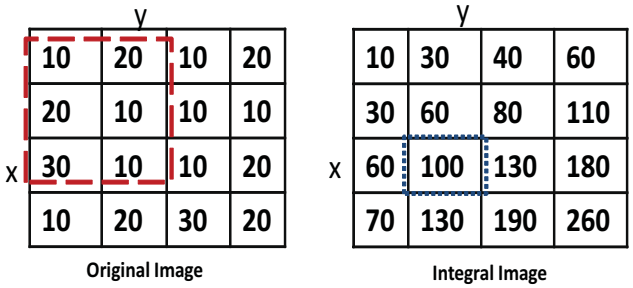

Figure 3: *Integral Image Representation*

stages of the cascade classifier created above. The value of any given feature is always the sum of the pixels within clear rectangles subtracted from the sum of the pixels within shaded rectangles. The evaluated image regions are sorted out between positive and negative images (i.e. vehicles and non-vehicles).

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. A positive rectangular features from a simple classifier is further evaluated by a second complex classifier from the cascade classifier. The detection process rejects many of the negative rectangular features and detects all of the positive rectangular features. This process continues for all the classifiers in the cascade classifier. This evaluation is time and resource consuming task and needed further optimization (Section V-B).

V. Experimental Setup & Results

This section explains the experimental setup used for generating the reported results and provides a detailed discussion of these results.

Video streams are acquired from the traffic monitoring cameras installed on roads/motorways using the Stream Acquisition component and are stored in the Storage Server. The processing server is deployed on compute nodes. The compute nodes have Intel Core i7 processor with 12 GB of RAM. The Storage Server and the APS server are connected through 1/10 Gbps LAN connection. The results from CPU compute node are compared with the results obtained from the GPU cluster (Section V-A).

We used OpenCV [25], image/video processing library, with C/C++ interface and its GPU component for implementing the analysis algorithms explained in Section IV. Some primitive image operations like converting image color space, image thresholding and image masking are used from OpenCV library in addition to HaarCascade Classifier algorithm (Section IV).

A. GPU Cluster

We explain the structure of our GPU cluster and different challenges faced while porting algorithms from CPU to GPU. A GPU cluster contains a host CPU node, a number of homogeneous/heterogeneous GPUs on PCI-Express interface. The compute nodes used in these experiments has 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 2496 processing cores. Nvidia Quadro 600 has 1 GB DDR3 RAM, 25.6 GBytes/sec data transfer rate, 2 multiprocessor units and 96 processing cores.

CUDA is used for implementing and executing the compute intensive parts of the vehicle detection algorithm on a GPU. It is an SDK, a software stack, uses SIMD (Single Instruction Multiple Data) parallel programming model, provides fine-grained data parallelism and thread parallelism nested within

Video Format	Video Stream Resolution	CPU Frame Buffer Time	Frame Transfer Time (CPU-GPU)	Single Frame Process Time		Total Video Stream Analysis Time	
				CPU	GPU	CPU	GPU
QCIF	177 X 144	0.11 msec	0.02 msec	3.03 msec	1.09 msec	9.39 sec	3.65 sec
CIF	352 X 288	0.28 msec	0.12 msec	9.49 msec	4.17 msec	29.31 sec	13.71 sec
4CIF	704 X 576	0.62 msec	0.59 msec	34.28 msec	10.17 msec	104.69 sec	34.13 sec
Full HD	1920 X 1080	2.78 msec	0.89 msec	44.79 msec	30.38 msec	142.71 sec	105.14 sec

Table III: *Single Video Stream Processing Time on CPU & GPU for the Supported Video Formats*

coarse-grained data and task parallelism [26]. A CUDA program starts its execution on CPU (called host), processes the data with CUDA kernels on a GPU (called device) and transfers the results back to the host [27].

Challenges in porting CPU application to GPU cluster

Main challenge in porting a host application (CPU based application) to a CUDA program is in identifying parts of the host application that can be executed in parallel and isolating data to be used by the parallel parts of the application. After porting the parallel parts of the host application to CUDA kernels, the program and data are transferred to the GPU memory and the processing results are transferred back to the host with the CUDA API function calls.

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.

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 2 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 (all the threads in same multiprocessor access consecutive memory locations) than that of non-coalesced memory access.

The above challenges are taken into account while porting our CPU application to the GPU cluster. The way, we tackled these challenges is detailed in Section V-B.

B. What is ported on GPU in our implementation and Why

A video stream consists of individual video frames. All of these video frames are independent of each other from vehicle 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 execution of HaarCascade Classifier algorithm, for detecting vehicles from the video streams, 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 in HaarCascade Classifier algorithm and consumed most of the processing resources and time. These functions are ported to GPU by writing CUDA kernels in our GPU implementation.

	# of Cars	Hit Rate	Miss Rate	Boosting Stages
Single-Scale Cars	200	88%	12%	12
Multi-Scale Cars	139	56%	44%	12

Table IV: *Classifier Performance*

The vehicle detection process executes partially on CPU and partially on GPU. In our case, the CPU decodes a video stream and extracts video frames from it. These video frames and cascade classifier data are ported to a GPU for vehicle detection. The CUDA kernels process a video frame and the vehicle detection results are transferred back to the CPU.

C. Experimental Results

We first present a performance evaluation of the trained cascade classifier obtained by following the process for creating a cascade classifier detailed in Section IV. The remainder of this section presents video stream analysis results.

Cascade Classifiers Performance

The UIUC image database [28] is used for creating the cascade classifier. We used a utility provided with OpenCV [25] for training the cascade classifier.

The images in this database are grey 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. 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. Minimum detection rate was set to 0.999 and 0.5 was set as maximum false positive rate. Test images data set varied in lightening conditions and in background scheme.

Performance of the trained classifier is 88% for the single-scale car images data set. Performance of the trained classifier was 56% for the mixed-scale car images data set. Since the classifier was trained with single-scale car images data set, less performance of the classifier with multi-scale car images was expected. Best detection results were found with 12 boosting stages of the classifier. The performance results of cascade classifier training are summarized in Table IV.

Video Stream Analysis

The presented framework is tested for analyzing the video streams for detecting vehicles. By analysis of a video stream, we mean decoding the video stream and detecting vehicles from it. The results presented in this paper focus on the processing time required for analyzing video streams. The analysis time of the recorded video streams using a CPU (Intel Core i7, 12GB RAM) is compared with the performance gains on a GPU cluster having Nvidia Tesla K20C and Nvidia Quadro 600 GPUs. The analysis of a video stream can be broken down into the following four steps:

Video Stream Length	CPU Analysis Time			GPU Analysis Time		
	Days	Hours	Seconds	Days	Hours	Seconds
2 Minutes	0.00	0.00	104.69	0.00	0.00	34.13
1 Hour	0.01	0.29	1046.9	0.00	0.03	255.96
1 Day	0.29	6.98	25125.6	0.03	0.64	6143.4
1 Week	2.04	48.86	175879.2	0.19	4.48	43003.8
1 Month	8.72	209.38	753768	2.13	51.2	184302

Table V: *Time for Analyzing Video Streams of Different Duration in 4CIF Video Format*

1) Decoding a video stream
2) Transferring a video frame and the classifier from CPU memory to the GPU memory
3) Processing a video frame data on a GPU/CPU
4) Downloading the results from GPU to CPU

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

Decoding a Video Stream

Video stream analysis is started by decoding a video stream using FFmpeg library. It involves reading a video file from the hard disk 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 complete video stream 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.

Video stream decoding took between 0.11 to 2.78 milliseconds for decoding an individual video frame from a video stream. The total time for decoding a video stream varied between 330 milliseconds to 8.34 seconds for the supported video formats. It can be observed from Figure 4a 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.

Transfer Video Frame and Classifier Data from CPU to GPU

A 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 5GB 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 the video analytics.

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 time transfer from CPU to GPU, for a QCIF video stream took only 60 milliseconds and 2.67 seconds for a Full HD video stream. Transferring processed data back to CPU memory from GPU memory took almost the same time. Time taken to transfer a video frame and required data, for the supported formats, from CPU to GPU is summarized in Table III. No data transfer is required for the CPU implementation as CPU processes a video frame directly from the CPU memory.

Processing Video Frame Data on GPU/CPU

The processing of data for vehicle 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. Processing of an individual video frame means processing all of its pixels for detecting vehicles from it using the cascade classifier algorithm as detailed in Section IV. 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 3.27 seconds to 91.14 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. While processing a video frame, the CPU is also busy in executing crucial OS level processes 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 III summarizes the individual video frame processing time for the supported video formats on the CPU and on the GPU. An individual video frame reading time, the transfer time from CPU-GPU and the processing time on the CPU and on the GPU is graphically depicted in Figure 4a.

Single Video Stream Analysis Time

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

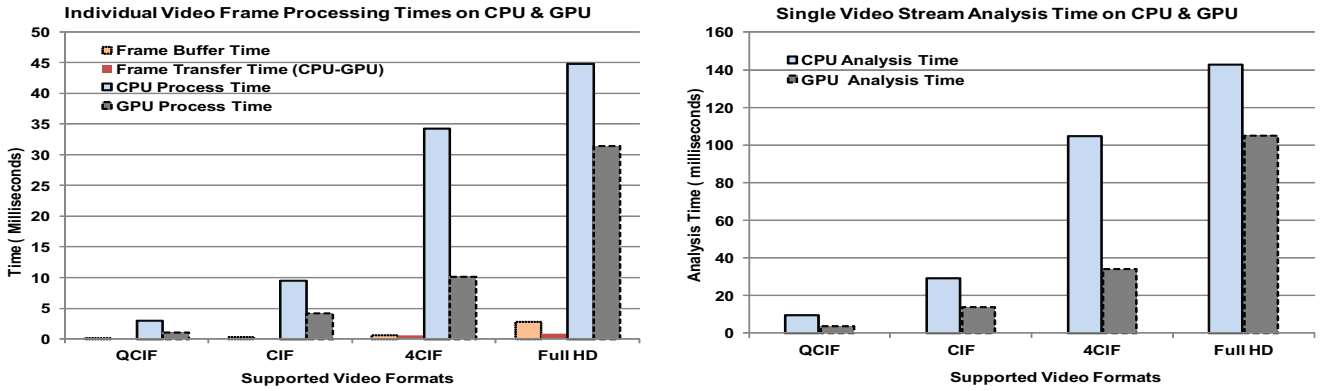

Figure 4: *(a) Frame Buffer, Transfer and Process Times for the Supported Video Formats, (b) Total Analysis Time of One Video Stream for the Supported Video Formats on CPU & GPU*

Full HD video stream of the same duration is 105.14 seconds. We observe a speed up between 1.35 times to 3.07 times for the supported video formats as compared to our CPU implementation.

The total video stream analysis time on CPU includes the video stream decoding time and the video stream 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 Full HD video streams on the CPU is slower and is taking more time than the length of a Full HD video stream. The video stream processing is a compute and data intensive task. Each video stream took 25% of the CPU processing power. 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 4CIF 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 Figure 4. The analysis time of 4CIF video streams from 2 minutes (minimum video file duration in our framework) to one month duration on a CPU and a GPU is summarized in Table V. Full HD video format has highest resolution and took more time to transfer video frame data from a CPU to the GPU and to process it. It showed least speed up as compared to our CPU implementation. Table III summarizes video stream processing time for the supported video formats.

		QCIF	CIF	4CIF	Full HD
GPU	Hours	5.47	20.57	51.2	15771.3
	Days	0.23	0.86	2.13	6.57
CPU	Hours	18.78	58.63	209.39	285.41
	Days	0.78	2.44	8.72	11.89

Table VI: *Time for Analyzing One Month of Recorded Video Streams data for the Supported Video Formats*

Parallel Analysis of Video Streams

We can analyse more video streams by processing them in parallel. As mentioned above, we could only analyse 3 video streams in parallel on a single CPU and were constrained by the availability of CPU processing power. We spawned multiple video streams processing threads from CPU to GPU. In this way, multiple video streams are processed in parallel on a GPU. The video frames of each video stream were processed sequentially in its own thread. We analyzed four parallel video streams on the GPU cluster having Tesla K20 and Quadro 600 GPUs, each analyzing 2 video streams in parallel. The time taken to analyse one month of recorded video stream data of the supported video formats on CPU and GPU is shown in Figure 5. 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 traffic video streams. The processing times for one month of recorded video stream data on a CPU and a GPU for the supported video formats is summarized in Table VI.

A human operator working continuously for 8 hours/day, without any breaks, would require three months to analyze one month of recorded video streams as compared to 6.57 days or 157.71 hours for Full HD video format on our GPU cluster. Analysis of the same data on a CPU requires 285.41 hours or 11.89 days for Full HD video format.

As mentioned earlier, the video streams from traffic monitoring cameras are usually recorded in CIF/4CIF formats. Analysis of one month of the recorded video streams in 4CIF format requires 51.20 hours (just over two days) on a GPU. This is a huge performance gain when compared with CPU execution that took about 162.40 hours. This speed up is 14 times faster than a human operator and 4 times faster than the CPU execution on one compute node. We expect more performance gain when analysis is performed in a cloud deployment with unlimited compute resources or on a GPU cluster in a cloud deployment. A comparison of CPU and GPU analysis times of one month of recorded video streams for the supported video formats is summarized in Table VI and is

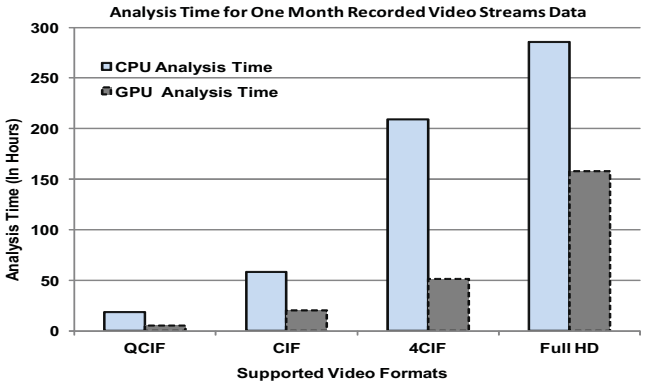

Figure 5: *Analysis Time of One Month of Video Streams Data on CPU & GPU for the Supported Video Formats*

graphically depicted in Figure 5.

VI. Conclusions & Future Research Directions

We have presented a framework for stream processing in clouds capable of detecting vehicles from the recorded video streams. The results of our implementation on a GPU cluster with two GPUs showed performance gain of 14 times when compared with one human operator doing the same analysis. CPU implementation yielded 4 times of analysis time improvement. We expect the video stream processing time to reduce further when the framework is ported onto a GPU cluster in a cloud.

It is important to mention that we are unable to use all the GPU resources in the above reported results, while executing multiple analytics processing threads on a single GPU. We are only using 15-20% of the GPU compute resources and 500 MB of the available GPU memory (5GB on Tesla K20). One possible reason for the less GPU load can be our way of spanning multiple threads from CPU to GPU and thread handling inside the GPU.

In future, we intend to address the software implementation bottlenecks for utilizing the maximum available resources of a GPU and will deploy the framework to a production cloud infrastructure. We will employ a feedback loop for the feature selection process in combination with artificial neural networks or semi supervised machine learning approaches for improving vehicle detection.

We would also extend our framework by making it more subjective. It will enable us to perform logical queries like, "How many cars of a specific color passed yesterday" from video streams. More sophisticated queries like, "How many cars of a specific color entered into the parking lot between 9 AM to 5 PM on a specific date" will also be included.

VII. Acknowledgments

This research is jointly supported by Technology Support Board, UK and XAD Communications, Bristol under Knowledge Transfer Partnership grant number KTP008832. The authors would like to say thanks to Gokhan Koch and Tulasi Vamshi Mohan from the video group of XAD Communications for their support in developing the software platform used in this research. We would also like to thank Erol Kantardzhiev and Dr. Ahsan Ikram from the data group of XAD Communications for their support in developing the file client library. We are specially thankful to Erol for his expert opinion and continued support in resolving all the network related issues.

References

[1] "The picture in not clear: How many surveliance cameras are there in the UK?" Research Report, July 2013.
[2] K. Ball, D. Lyon, D. M. Wood, C. Norris, and C. Raab, "A report on the surveillance society," Report, September 2006.
[3] M. Gill and A. Spriggs, "Assessing the impact of CCTV," London Home Office Research, Development and Statistics Directorate, February 2005.
[4] J. S. Bae and T. L. Song, "Image tracking algorithm using template matching and PSNF-m," *International Journal of Control, Automation, and Systems*, vol. 6, no. 3, pp. 413–423, June 2008.
[5] K. F. MacDorman, H. Nobuta, S. Koizumi, and H. Ishiguro, "Memory-based attention control for activity recognition at a subway station," *IEEE MultiMedia*, vol. 14, no. 2, pp. 38–49, April 2007.
[6] C. Stauffer and W. E. L. Grimson, "Learning patterns of activity using real-time tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 747–757, August 2000.
[7] C. Stauffer and W. Grimson, "Adaptive background mixture models for real-time tracking," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1999, pp. 246–252.
[8] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2001, pp. 511–518.
[9] S. Mantri and D. Bullock, "Analysis of feedforward-back propagation neural networks used in vehicle detection," *Transportation Research Part C– Emerging Technologies*, vol. 3, no. 3, pp. 161–174, June 1995.
[10] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine Learning*, vol. 37, no. 3, pp. 297 – 336, December 1999.
[11] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *26th IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, 2010.
[12] A. Ishii and T. Suzumura, "Elastic stream computing with clouds," in *4th IEEE Intl. Conference on Cloud Computing*, 2011, pp. 195–202.
[13] Y. Wu, C. Wu, B. Li, X. Qiu, and F. Lau, "CloudMedia: When cloud on demand meets video on demand," in *31st International Conference on Distributed Computing Systems*, 2011, pp. 268–277.
[14] J. Feng, P. Wen, J. Liu, and H. Li, "Elastic Stream Cloud (ESC): A stream-oriented cloud computing platform for rich internet application," in *Intl. Conf. on High Performance Computing and Simulation*, 2010.
[15] "Vi-system," http://www.agentvi.com/.
[16] "Smartcctv," http://www.smartcctvltd.com/.
[17] "Project BESAFE," http://imagelab.ing.unimore.it/besafe/.
[18] B. S. System, "IVA 5.60 intelligent video analysis," Bosh Security System, Tech. Rep., 2014.
[19] "EPTACloud," http://www.eptascape.com/products/eptaCloud.html
[20] "Inteligent vision," http://www.intelli-vision.com/products/intelligent-video-analytics.
[21] K. yan Liu, T. Zhang, and L. Wang, "A new parallel video understanding and retrieval system," in *IEEE Intlernational Conference on Multimedia and Expo (ICME)*, July 2010, pp. 679–684.
[22] H. Schulzrinne, A. Rao, and R. Lanphier, "Real time streaming protocol (RTSP)," Internet RFC 2326, April 1996.
[23] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," Internet RFC 3550, 2203.
[24] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, July 2003.
[25] "Opencv," http://opencv.org/ .
[26] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," *Queue-GPU Computing*, vol. 16, no. 2, pp. 40–53, April 2008.
[27] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*, 1st ed. Addison-Wesley Professional, 2010.
[28] http://cogcomp.cs.illinois.edu/Data/Car/.